THE BEST SOFTWARE WRITING I

Selected and Introduced by

Joel Spolsky

Apress®

The Best Software Writing I: Selected and Introduced by Joel Spolsky

ISBN (pbk): 1-59059-500-9

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editor: Gary Cornell
Editorial Board: Steve Anglin, Dan Appleman, Ewan Buckingham, Gary Cornell, Tony Davis, Jason Gilmore, Jonathan Hassell, Chris Mills, Dominic Shakeshaft, Jim Sumser
Assistant Publisher: Grace Wong
Project Manager: Beth Christmas
Copy Edit Manager: Nicole LeClerc
Copy Editor: Liz Welch
Production Manager: Kari Brooks-Copony
Production Editor: Kelly Winquist
Compositor: Dina Quan
Proofreader: Nancy Sixsmith
Indexer: Broccoli Information Management
Cover Designer: Kurt Krames
Manufacturing Manager: Tom Debolski
Licensing: Tina Nielsen

Distributed to the book trade in the United States by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013, and outside the United States by Springer-Verlag GmbH & Co. KG, Tiergartenstr. 17, 69112 Heidelberg, Germany.

In the United States: phone 1-800-SPRINGER, fax 201-348-4505, e-mail orders@springer-ny.com, or visit http://www.springer-ny.com. Outside the United States: fax +49 6221 345229, e-mail orders@springer.de, or visit http://www.springer.de.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail info@apress.com, or visit http://www.apress.com.

To my sister Ruth, כִּי יוֹדֵעַ כָּל-שֵׁשׁ עַמִּי, כִּי אֵשֶׁת חַיִל אָתְּ

CONTENTS

ABOUT THE EDITOR

Joel Spolsky is a globally recognized expert on the software development process. His website, *Joel on Software* (`www.joelonsoftware.com`), is popular with software developers around the world and has been translated into over 30 languages. As the founder of Fog Creek Software in New York City, he created FogBugz, a popular project management system for software teams. Joel has worked at Microsoft, where he designed VBA as a member of the Excel team, and at Juno Online Services, developing an Internet client used by millions. He has written two previous books: *User Interface Design for Programmers* (Apress, 2001) and *Joel on Software* (Apress, 2004). Joel holds a BS from Yale in computer science. Before college he served in the Israeli Defense Forces as a paratrooper, and he was one of the founders of Kibbutz Hanaton.

ABOUT THE AUTHORS

Ken Arnold has loitered around the computing field for decades, including attending Berkeley where he worked on the BSD project, creating the `curses` library and helping on `rogue`; writing the "The C Advisor" column for *Unix Review* (later "The C++ Advisor" as progress led us down the garden path); coauthoring *The Java Programming Language* and other books; designing JavaSpaces and helping design Jini; and occasionally (as shown here) pretending to be hip by blogging. His current dalliances include the human factors of programming languages and APIs, electronic voting systems your mother could trust, and the Napkin pluggable look and feel for Java that makes provisional GUIs look, well, provisional.

Leon Bambrick is a prolific programmer, satirist, and pugilist, working out of the southern hemisphere. He first met Joel Spolsky when they were stranded together on a desert island, with nothing but an 8086 and a copy of Kernigan and Ritchie. His website, `secretGeek.net`, has a small cameo in *Star Wars Episode III – Revenge of the Sith*—as an Imperial Guard's codpiece.

Michael Bean is a software developer and entrepreneur. He is currently president and one of the founders of Forio Business Simulations. Before Forio, Michael held senior management posts at consulting and software firms in the United States and Europe. Michael was also a research associate for the System Dynamics Group at MIT, where he developed simulations that analyzed the strategic implications of manager decisions. Michael has consulted with corporations and government agencies nationally and internationally on transfer pricing, competitive strategy, emerging technologies, and customer migration. He has conducted scenario planning, systems thinking, and computer simulation

seminars to corporations and government agencies worldwide. In addition, Michael has presented at national conferences on strategy, software, and computer simulation.

Rory Blyth works for Microsoft as a corporate stooge. In his spare time, he keeps a blog at `www.neopoleon.com`, ponders the universe, and considers himself to be one of the three missing Sankara stones, although he probably isn't, but it makes him feel better about being so bloody insignificant.

Adam Bosworth joined Google recently as vice president of engineering. He came to Google from BEA, where he was chief architect and senior VP of advanced development and responsible for driving the engineering efforts for BEA's Framework Division. Prior to joining BEA, he cofounded Crossgain, a software development firm acquired by BEA. Known as one of the pioneers of XML, he held various senior management positions at Microsoft, including general manager of the WebData group, a team focused on defining and driving XML strategy. While at Microsoft, he was responsible for designing and delivering the Microsoft Access PC Database product and assembling and driving the team that developed Internet Explorer 4.0's HTML engine.

danah boyd is a PhD student in the School of Information Management and Systems at the University of California, Berkeley, where she studies how people negotiate a presentation of self in mediated social contexts to unknown audiences using ethnographic methods. She is particularly interested in how youth develop a culturally situated understanding of self and the role of technology in this process. Prior to Berkeley, danah received a master's in sociable media from the MIT Media Lab and a bachelor's in computer science from Brown University. Her work has ranged from psychological studies of how depth cue prioritization is dependent on levels of sex hormones to design installations of interactive social visualizations. danah blogs extensively at Apophenia (`www.zephoria.org/thoughts`) and Many-to-Many (`www.corante.com/many`).

Raymond Chen has worked in Microsoft's Windows division since 1992 and has seen a lot of things come and go. His blog deals with the history of Windows and the dying art of Win32 programming.

Kevin Cheng is an independent user experience specialist and global nomad. He holds a master's in human-computer interaction and ergonomics from the University College London Interaction Centre (UCLIC) and has spoken at UXNet, UPA, and ACM-SIGCHI. He is the cofounder and cocreator of OK/Cancel (`www.ok-cancel.com`), an online site believed to be in the top five of usability and HCI-themed comics.

Tom Chi has a Masters Degree in Electrical Engineering, which probably means he's qualified neither to talk about HCI nor to write any sort of funny thing. Yet, week after week he dreams the impossible dream at `ok-cancel.com`. As for credentials, there is the small matter of having designed UI features for two releases of Microsoft Outlook, as well as his dark history of consulting for F500 clients—but these are topics that civilized people shouldn't speak of. Shhh.

Cory Doctorow (`craphound.com`) is European Affairs Coordinator for the Electronic Frontier Foundation (`www.eff.org`), a member-supported nonprofit group that works to uphold civil liberties values in technology law, policy, and standards. He represents EFF's interests at various standards bodies and consortia, and at the United Nations' World Intellectual Property Organization. Doctorow is also a prolific writer who appears on the mastheads at *Wired*, *Make*, and *Popular Science* magazines, and whose science fiction novels have won the Campbell, Sunburst, and Locus Awards. His novel *Down and Out in the Magic Kingdom* is a finalist for this year's Nebula Award. He is the coeditor of the popular weblog Boing Boing (`boingboing.net`). Born in Canada, he now lives in London, England.

Bruce Eckel (`www.BruceEckel.com`) is the author of *Thinking in Java* (Prentice Hall, 1998, 2nd edition, 2000, 3rd edition, 2003, 4th edition, 2005), the *Hands-On Java Seminar CD-ROM* (available on the website), *Thinking in C++* (PH 1995; 2nd edition 2000, Volume 2, with Chuck Allison, 2003), and *C++ Inside & Out* (Osborne/McGraw-Hill, 1993), among others. He's given hundreds of presentations throughout the world, published over 150 articles in numerous magazines, was a founding member of the ANSI/ISO C++ committee, and speaks regularly at conferences. He provides public and private seminars and design consulting in C++ and Java.

Paul Ford is an editor at *Harper's* magazine, a frequent commentator on NPR's *All Things Considered*, and the sole proprietor of Ftrain.com. He has fooled with computers for the last two decades, and feels no inclination to stop. He lives in Brooklyn, New York.

Paul Graham is an essayist, programmer, and programming language designer. In 1995 he developed with Robert Morris the first web-based application, Viaweb, which was acquired by Yahoo in 1998. In 2002 he described a simple Bayesian spam filter that inspired most current filters. He's currently working on a new programming language called Arc, a new book (probably) for O'Reilly, and is one of the partners in Y Combinator. Paul is the author of *On Lisp* (Prentice Hall, 1993), *ANSI Common Lisp* (Prentice Hall, 1995), and *Hackers & Painters* (O'Reilly, 2004). He has an AB from Cornell and a PhD in computer science from Harvard, and studied painting at RISD and the Accademia di Belle Arti in Florence.

John Gruber is a freelance writer, web developer, designer, and Mac nerd. He combines those interests on his website, Daring Fireball (`http://daringfireball.net/`). John lives in Philadelphia with his wife and son.

Gregor Hohpe leads the Enterprise Integration practice at ThoughtWorks, Inc., a specialized provider of application development and integration services. Gregor is a widely recognized thought leader on asynchronous messaging architectures and coauthor of the seminal book *Enterprise Integration Patterns* (Addison-Wesley, 2004). Gregor speaks regularly at technical conferences around the world and maintains the website `www.eaipatterns.com`.

Ron Jeffries has been developing software longer than most people have been alive. He holds advanced degrees in mathematics and computer science, both earned before negative integers had been invented. His teams have built operating systems, compilers, relational database systems, and a large range of applications. Ron's software products have produced revenue of over half a billion dollars, and he wonders why he didn't get any of it.

Eric Johnson graduated from the University of Illinois with a BS in computer science in 1993 and has worked at FactSet Research Systems ever since. Currently he is the director of market data engineering and lives with his wife and two kids in southwestern Connecticut. He can be reached at johnson.eric@gmail.com.

Eric Lippert has been a software developer at Microsoft since 1996. He spent his first five years working on VBScript, JScript, Windows Script Host, and other scripting technologies and more recently has been working on Visual Studio Tools For Office. He also writes a blog, where he dispenses advice about scripting, security, and (occasionally) romance. When not writing software or writing about software, Eric can be found playing old songs on old pianos, trying to keep the mast of his tiny sailboat upright, building kites, or talking his friends into helping him fix his 97-year-old house.

Michael "Rands" Lopp is a Silicon Valley–based software engineering manager. He's ridden a variety of high-tech roller-coasters, including Borland International, Netscape Communications, Apple Computer, and a start-up you've unfortunately never heard of. In his spare time, he writes a weblog at www.randsinrepose.com, where he optimistically contemplates the fact that the world continues to get uncomfortably smaller.

Larry Osterman has been working at Microsoft since 1984. In that time, he's worked as a software engineer deep in the plumbing of various Microsoft® products, including MS-DOS, MS-NET, LAN Manager, Windows NT, Exchange, and eHome, and is currently working in the Windows Multimedia Technologies group. Larry lives just north of Seattle with his wife Valorie and their two kids, four cats, and two horses.

Mary Poppendieck is a seasoned leader in both operations and new product development with more than 25 years of IT experience. She has led teams implementing lean solutions ranging from enterprise supply chain management to digital media, and built one of 3M's first just-in-time lean production systems. Mary is currently the president of Poppendieck LLC in Minnesota. Her book *Lean Software Development: An Agile Toolkit,* which brings lean principles to software development, won the Software Development Productivity Award in 2004.

Rick Schaut grew up in Green Bay and Milwaukee, Wisconsin, where he spent his childhood watching Paul Hornung score touchdowns and Hank Aaron hit home runs. At one point, he believed that our national anthem ended with, "the land of the free and the home of the Braves," and he had a hard time figuring out why every American League umpire was named "Al." After graduating from high school, Rick studied economics at the University of Wisconsin, Milwaukee, and computer science at the University of Wisconsin. Rick joined Microsoft in 1990, and has been working on versions of Microsoft Word ever since.

Clay Shirky teaches at NYU's graduate Interactive Telecommunications Program, and works with clients, including the Library of Congress, Connecting for Health, and Nokia, on network design issues. He writes about the cultural and economic issues of the Internet (archived at `shirky.com`).

Eric Sink is the founder of SourceGear, a developer tools ISV. More of Eric's writings and rants can be found on his weblog at `software.ericsink.com`. Eric and his wife live in central Illinois with their two young daughters and one old cat.

Aaron Swartz is a teenage writer, hacker, and activist. Formerly the Metadata Advisor to Creative Commons and member of the W3C's RDF Core Working Group, he is currently a student at Stanford University, where he authors his popular weblog and is beginning work on a technology startup.

why the lucky stiff is a computer progg'er and aspiring author with no true achievements under his belt. Except there was that time when he tore a building in half with his bare feet.

INTRODUCTION

New York City is a blast.

Just the other day, as I was walking the four blocks from my office to the subway entrance, interesting things kept happening.

Not *really* interesting things, just modestly interesting things.

So, for example, some guy was running down the sidewalk frantically, looking very much like a character in an R. Crumb comic, flapping his arms broadly and making chicken sounds. Running isn't the right word. He was kind of pratfalling repeatedly and then catching himself right before he hit the ground.

Then a taxi turning the corner nearly knocked over an old man who was crossing the street a little bit too slowly for the taxi driver's taste.

A couple of chubby, red-faced out-of-towners asked me if there was a bar anywhere nearby. (There was. We were in front of it.)

Someone was handing out little advertising cards at the entrance to the subway. Of course, the inside of the subway station was completely *littered* with the cards because everybody who took one immediately hurled it on the ground as violently as you can hurl a four-by-six postcard. I almost slipped on one on the steps down.

Modestly interesting stuff, but quite forgettable in New York.

The next day I was talking to one of the summer interns we just hired. For some reason, this year's summer intern class consists of 75% people who are either from Indiana or who went to school in Indiana. Indiana, for those of you not familiar with our American landscape, is somewhere in the middle—a state of farms, wholesome colleges with corn-fed basketball-playing kids, Norman Rockwell towns, and the occasional rust-belt hellmouth industrial city gasping its last breath. (As I write these words I brace for the slew of angry letters from the Indiana Department of Tourism and Infrastructure promoting the exciting cultural scene, the many picturesque lakes, the world-class telephone

system, and the variety of ethnic restaurants. You might find a Mexican restaurant and an Italian restaurant on the same block!)

Anyway, the intern said he had never lived in New York City, and asked me what it was like. I didn't really have a good answer, but I said, "New York is the kind of place where 10 things happen to you every day *on the way to the subway* that would have qualified as interesting dinner conversation in Bloomington, Indiana, and you don't pay them any notice."

Feeling smug with myself, I pulled down an atlas from the bookshelf to find another state to insult.

*Any*how, I can't remember why I told you that story.

Oh, wait, yes I can, but first I have to tell you *another* story.

A few months ago, I got a review copy of a book from another publisher, other than the publisher of this book, who will remain anonymous, and the book will remain anonymous, and the author will remain anonymous, because I'm afraid I just have *nothing* good to say about said book.

The publisher wanted to get a quote from me to put on the back cover talking about how wonderful his book was. Normally I'd be happy to do that; I'm a complete publicity slut and will do just about anything to get my name in front of the reading public. My hope is that if I do this enough, telemarketers who call me at home will be able to pronounce my name.

The book started out looking promising. It filled a real need. I remember several times standing in bookstores desperately trying to find a book on the very topic, but there was nothing to be found. So I started reading the manuscript full of high hopes.

Bleah.

I could hardly bear to keep reading.

The author kept saying smart and interesting things.

He even wrote clearly.

But the book was thoroughly, completely, *boring*. And worse, it was completely unconvincing.

The author had violated the number one rule of good writing, the "Show, don't tell" rule. There was not a single story in the book. It was chock-full of sentences like "A good team leader provides inspiration by setting a positive example." What the eff?

Pay attention. Here's the way to say "A good team leader provides inspiration by setting a positive example" without putting your audience to sleep:

> For a few months in the army I worked in the mess hall, clearing tables and washing dishes nonstop for 16 hours a day, with only a half-hour break in the afternoon, if you washed the dishes really fast. My hands were permanently red, the front of my shirt was permanently wet and smelly, and I couldn't take it any more.
>
> Somehow, I managed to get out of the mess hall into a job working for a high-ranking Sergeant Major. This guy had years of experience. He was probably 20 years older than the kids in the unit. Even in the field, he was always immaculate, wearing a spotless, starched, pressed full dress uniform with impeccably polished shoes no matter how dusty and muddy the rest of the world was around him. You got the feeling that he slept in 300-threadcount Egyptian cotton sheets while we slept in dusty sleeping bags on the ground.
>
> His job consisted of two things: discipline and the physical infrastructure of the base. He was a bit of a terror to everyone in the battalion due to his role as the chief disciplinary officer. Most people only knew him from strutting around the base conducting inspections, screaming at the top of his lungs and demanding impossibly high standards of order and cleanliness in what was essentially a bunch of tents in the middle of the desert, alternately dust-choked or mud-choked, depending on the rain situation.
>
> Anyway, on the first day working for the Sergeant Major, I didn't know what to expect. I was sure it was going to be terrifying, but it *had* to be better than washing dishes and clearing tables all day long (and it's not like the guy in charge of the mess hall was such a sweetheart, either!).
>
> On the first day he took me to the officers' bathroom and told me I would be responsible for keeping it clean. "Here's how you clean a toilet," he said.
>
> And he got down on his knees in front of the porcelain bowl, in his pressed starched spotless dress uniform, and scrubbed the toilet with his bare hands.
>
> To a 19-year-old who has to clean toilets, something which is almost *by definition* the worst possible job in the world, the sight of this high-ranking, 38-year-old, immaculate, manicured, pampered discipline officer cleaning a toilet completely reset my attitude. If he can clean a toilet, I can clean a toilet. There's nothing wrong with cleaning toilets. My loyalty and inspiration from that moment on were unflagging. *That's* leadership.

See what I did here? I told a story. I'll bet you'd rather sit through 10 of those 400-word stories than have to listen to someone drone on about how "a good team leader provides inspiration by setting a positive example."

Anyway, I called up the editor of the book that they wanted me to praise, and said I couldn't, in good faith, recommend a boring book without any stories in it, even if it was 100% correct and otherwise well-written. I think they hate me now.

So be it.

The software development world desperately needs better writing. If I have to read another 2000-page book about some class library written by 16 *separate* people in broken ESL, I'm going to flip out. If I see another hardback book about object-oriented models written with dense faux-academic pretentiousness, I'm not going to shelve it any more in the Fog Creek library: it's going right in the recycle bin. If I have to read another spirited attack on Microsoft's buggy code by an enthusiastic nine-year-old Trekkie on Slashdot, I might just poke my eyes out with a sharpened pencil. Stop it, stop it, stop it!

And that's why when Gary Cornell suggested this book, I leapt at the idea. It would be a chance to showcase some of the best writing about software from the past year "or so." The original idea was to make it an annual, so the volume you're holding would be "The Best Software Writing of 2004," but there were a bunch of great articles from 2003 that we wanted to include, and we were afraid bookstores would return it at the end of the year if there was a date in the title. I solicited nominations from the faithful readers of my website, *Joel on Software,* and selected the final stories myself, so the blame for what's included and what isn't included is entirely my own, but full credit for really incredible writing in a field that doesn't normally get any goes to the contributors.

Ken Arnold

STYLE IS SUBSTANCE[1]

Python did something really interesting: it made whitespace matter for the first time in a major programming language since FORTRAN on punched cards. In Python, the way you create a block is not by surrounding it with begin and end or { and }, but by indenting it. That's all.

A lot of geeks instinctively cringed. "Whitespace should never matter!" they claimed, without remembering why. The main reason was that you couldn't always see whitespace, because it's, um, white. So, for instance, in the standard Unix Make, where certain lines must begin with a tab character, if you or your editor replaced such a tab character with eight spaces (how helpful!), you would suddenly find your makefile didn't work, and without any explanation. So we all learned: whitespace mustn't matter!

Well, yeah.

Maybe we went too far.

Here's the beauty of Python.

In C-like languages (C, C++, Java) the human eye sees indentation as defining a block, but the compiler sees the { and the }. So in cases where the indentation and the braces disagree, the one that is less visible to humans—the braces—wins out. But why do we need two ways to indicate a block, one for humans and one for compilers? Why not stick with one way, so code always looks like what it does?

1. Ken Arnold, "Style Is Substance," Artima Weblogs, Notes from Underfoot (`http://www.artima.com/weblogs/index.jsp?blogger=arnold`), October 6, 2004. See `http://www.artima.com/weblogs/viewpost.jsp?thread=74230`.

Ken Arnold took this small idea from Python all the way. He proposes something even more radical—which, like many great ideas, is so crazy it just might work. – Ed.

. . . wherein I decide to wade into the programming language equivalent of TV wrestling: coding style . . .

I'm sure this will cause me no end of grief, but I'm about to confess publicly here that I am a heretic. (In this particular case I'm only confessing to heresy in computer language design. Other heresy confessions will have to await another time.)

I'll state it right out: For almost any mature language (C, Java, C++, Python, Lisp, Ada, FORTRAN, Smalltalk, sh, JavaScript, etc.) coding style is an essentially solved problem, and we ought to stop worrying about it. And to stop worrying about it will require worrying about it a lot at first, because the only way to get from where we are to a place where we stop worrying about style is to enforce it *as part of the language*.

Yup. I'm really saying that. I'm saying that, for example, the next ANSI C update should define the standard K&R C programming style[2] into the language grammar. Programs that use any new features should be required to be in K&R style or be rejected by the compiler as syntactically illegal.

I'm gonna pause here. When I was talking about this on a mailing list I had to go through this several times. People didn't quite get me because they couldn't believe someone was saying this. I mean this literally. For example, I want the next C grammar to define that a space comes between any keyword and an opening parenthesis: `if (foo)` would be legal, but `if(foo)` would not. Not a warning, not optionally checked, but actually forbidden by the language parser. Flat-out illegal. Can't compile.

2. From Brian Kernighan and Dennis Ritchie's *The C Programming Language* (Prentice Hall, 1988), the standard and founding tome for the language.

Here is the logic in its simplest form:

- **Premise 1: For any given language, there are one or a few common coding styles.**

  Typically one is set by the founder(s) or earliest documenter, but others will evolve over time. But even for C there are only a handful of commonly used styles, ignoring trivial variations.

- **Premise 2: There is not now, nor will there ever be, a programming style whose benefit is significantly greater than any of the common styles.**

  Get real. Discovering a style that improves your productivity or code quality by more than a few percent over the common styles is about as likely as discovering a new position for sex. (Astronauts need not apply, unless they want to invite me along.)

- **Premise 3: Approximately a gaboozillion cycles are spent on dealing with coding style variations.**

  Think about it: How many reformatter/pretty-printers projects are there on SourceForge[3] alone? How many options does any given IDE (including emacs) have for formatting code? How many cycles are spent deciding on a style, documenting it, enforcing it, and updating it? How many history logs for CVS, ClearCase, etc., have a lot of noise from varying format changes? How many brain cycles are spent on arguing about this topic?

- **Premise 4: For any nontrivial project, a common coding style is a good thing.**

  I really think this is pretty well agreed on. How constraining the style is varies, but having several folks hacking on the same code with conflicting coding styles introduces more pain than any single style imposes on any single person. Every project I know of has a style, if not spelled out at least by custom.

- **Conclusion: Thinking of all the code in the entire world as a single "project" with a single style, we would get more value than we do by allowing for variations in style.**

3. See `http://sf.net`.

Think of it. All the programming examples in one style. Web pages, journals, papers, emails use one style. Reformatting issues gone. Arguments over whose style is better gone. Reformatters become a quaint historical artifact.

And most of all: *No More Style Wars!* Really! Think of all those cycles that we could then plow into something more productive, like vi/emacs wars! Or world peace! Or a *really* good chocolate cookie recipe! You choose!

Of course, you will never enforce any style globally unless people have literally no choice. How many C programmers use "during" as a stylistic preference to the keyword "while"? (Preprocessor abusers need not apply. On second thought, please do: We need to identify you for our eugenics program.) Or skip the parentheses around an `if` clause? They don't because they *can't*. You know some would if they could. The thing that stops these "personal styles" is that the C compiler will not accept them. If you can't compile your code you fix it. It's so simple it's stupid. And therefore it works.

So I want the owners of language standards to take this up. I want the next version of these languages to require any code that uses new features to conform to some style. Let the standards committees gnash and snarl and wring their hands over which of the common styles is the winner. Sell tickets. We all get to comment and the language standards geeks decide. We know where they'll go—C will go to K&R; C++ will go with Bjarne's style (excuse me while I cringe); Java will go with the Sun style as shown in the language spec and most of the Java books from Sun (including mine); Lisp style is almost already set mostly in stone. Perl is a vast swamp of lexical and syntactic swill and nobody knows how to format even their own code well, but it's the only major language I can think of (with the possible exception of the recent, yet very Java-like C#) that doesn't have at least one style that's good enough.

Some things are either uncheckable (Hungarian notation, using "get" and "set" method prefixes) or not widely agreed upon (such as import/#include ordering). These can be left for future standards. Or not. The owners of the standard decide. But whatever they do, they should set the style and build it into the actual freakin' grammar.

This heresy encompasses one major sub-heresy: That whitespace should matter.

Most style rules have to do with the placement of whitespace: newlines before or after curly braces, whitespace around operators or not,

etc. So I'm saying that languages should indeed care about whitespace. A lot.

Yet one of the things we supposedly learned from languages like FORTRAN was that whitespace should only matter to mark boundaries between tokens. This was accepted wisdom because FORTRAN had columns—the first five columns were reserved for a statement number or a comment indicator, the sixth column with any character in it meant a continuation of the previous line, seven through 72 had the code, and the last eight were reserved for sequence numbers useful for reordering the card deck if it was dropped. Yes, I mean cards, the physical type, with rectangular holes. So if you put something in the wrong column, a statement could become a comment or whatever, which was really annoying. Also, `DO10I=1,100` was the same as `DO 10 I = 1, 100` because `DO` was a keyword followed by a number and so the space wasn't required, although it made `DO10I=1` interesting, as that assigned `1` to a variable named `DO10I`.

I lived this ugliness, so I felt the pain. But this didn't prove that whitespace shouldn't matter. All it really proved was that FORTRAN's whitespace rules sucked. Freedom to put whitespace anywhere has proven to be expensive and cycle-wasting in practice. We're not editing on punched cards anymore, and reformatters are as common as spam. We can use this power: type code however you want to but before you compile it, reformat it (or reformat on the fly, whatever).

In the end, this requires only that editors and IDEs will let you type stuff and make it look right. This is basically just reformatting on the fly, which many editors already do. We don't need you to type zero, one, or 17 spaces between an `if` and its open parenthesis, we just need the editor (assuming K&R C style) to put exactly one space there. And getting even this right will be easier if there is only one style to worry about. It's one of the things that those reformatting or style-adapting cycles can go to.

Basically, freedom for formatting style has proven extremely expensive, and does not deliver much value for cost. Think of it this way: could you honestly fill in the following:

> I, *[insert name here],* know of a programming style whose impact on programmer productivity and/or program quality is large enough that my freedom to choose it over any major common style validates the programmer productivity and investment used industry wide in arguing about style, imposing style, and reformatting to match styles. That style is *[insert style description here]* and its benefits are *[insert benefits here]*.

Or even the less demanding:

> I, *[insert name here]*, know of a programming style whose impact on programmer productivity and/or program quality is ≥ 5% when compared to any major common style. That style is *[insert style description here]* and its benefits are *[insert benefits here]*.

I think you will mostly get snickers even suggesting that this can be filled out. And on a single project alone you can spend 5% on coding style issues—mostly up front, but it's a continuous bleeding: style wars cropping up over things as yet undefined; new tools being suggested, written, or integrated; people forgetting to put it in the right style getting corrected, polluting the change history; training new people in the style; disciplining engineers who are uncooperative; and general bitching, whining, and moaning.

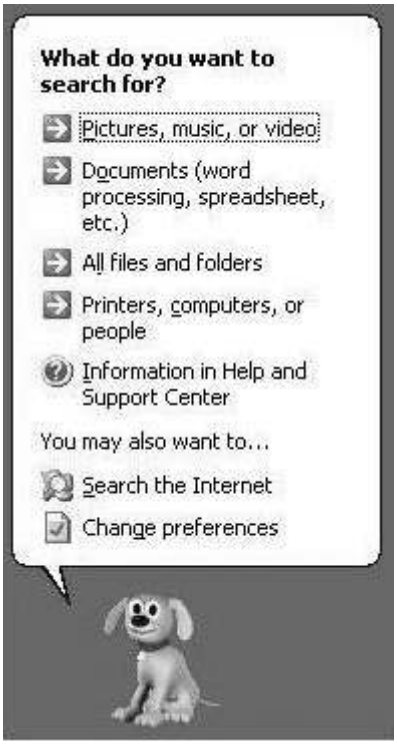
So 5% doesn't even touch the opportunity costs and other pain associated with not having a mandated style across all the code in the world.

Or if you prefer the question the other way 'round: What benefits do we get from freedom of style that outweighs the cost we pay for it?

To me the answer seems obvious: nowhere *near* enough.

Leon Bambrick

AWARD FOR THE SILLIEST USER INTERFACE: WINDOWS SEARCH[1]



Why is a dog asking me questions?

Who's been putting the mescaline in the Microsoft Koolaid?

What if Google used this approach?

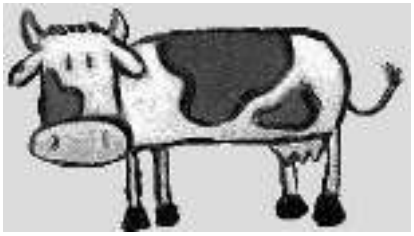
Would Google still be number one?

1. Leon Bambrick, "Award for the Silliest User Interface: Windows Search," secretGeek (http://www.secretgeek.net). See http://www.secretgeek.net/ms_search.asp#.

Google

So You'd Like to Search for Something!

- Do you know where you last saw it? Yes or No
- Is it bigger than a breadbox? Or smaller?
- Is it animal, mineral, or vegetable?
- Maybe you should buy a personal organizer! Then you won't keep losing things.
- Did you check under the bed?



©2004 Google - Searching 8,058,044,651 web pages

Unless Microsoft can correct this wrongheaded approach to search, WinFS promises to be so overengineered as to be completely unusable.