

Why Word is a terrible program

B Klemens

8 January 2006

First of all, it is time to speak some truth to power in this country:
Microsoft Word is a terrible program.

[. . . For example,] there is the moment when you realize that your notes are starting to appear in 12-pt. Courier New. Word, it seems, has, at some arbitrary point in the proceedings, decided that although you have been typing happily away in Times New Roman, you really want to be in the default font of the original document. You are confident that you can lick this thing: you painstakingly position your cursor in the Endnotes window (not the text!, where irreparable damage may occur) and click Edit, then the powerful Select All; you drag the arrow to Normal (praying that your finger doesn't lose contact with the mouse, in which case the window will disappear, and trying not to wonder what the difference between Normal and Clear Formatting might be) and then, in the little window to the right, to Times New Roman. You triumphantly click, and find that you are indeed back in Times New Roman but that all your italics have been removed. What about any of this can be considered high-speed?

From *The end matter* by Louis Menand, The New Yorker, issue of 2003-10-06.

If you are a casual user, then Word is probably fine for you—maybe even ideal. It's great that there is a product out there that will help complete novices and your proverbial Aunt Myrtle to produce beautiful documents.

But for many office workers, regardless of their job title, their actual occupation is "Word user". They come in at nine-ish in the morning, edit documents in Word for eight hours, and go home. If that describes you, if half

of your waking life is spent staring at that program, then you no doubt have a strong interest in working out how to use your most-used tool efficiently. Maybe the right tool for Aunt Myrtle is not the right tool for you, and your life would be better if you could jump ship for something better.

I realize that Word is a standard that many of you are forced to use by your employers or colleagues. After you send your boss or colleague a copy of this paper, you may want to have a look at Section 6 for practical suggestions on what you can do to remain compatible with your coworkers while not letting them drag you down into inefficiency.

If you don't have the attention-span for a 23-page paper on efficient software design, perhaps the key thing to take away is that there are indeed 23 pages worth of things to say about how inefficient Word is.

I recently wrote a book entitled *Math You Can't Use: Patents, Copyright, and Software*, so I should clarify that that book and this document are 100% unrelated. That book was about the law and politics of software, but this paper isn't: this paper is about efficiency and usability, and I promise you minimal discussion of the politics of Microsoft in the pages that follow.

By the way, my publisher asked me to write that book in Word, so I have ironic first-hand experience with using Word to write a complex document.

1 Semantic editing

The key failing of Word is the difficulty of semantically-oriented editing.

The way most of us format a document in a word processor is to change the formatting of individual elements as we need them. Titles need to be marked in boldface; there needs to be this much space put between paragraphs; the margins should be just so on the cover page. I will call this *literal markup*, where you make changes on the screen until the text looks the way you want it to look.

The alternative is to specify what each element actually *means*, and then worry about how the formatting happens later. Mark the titles as <title>, mark the paragraphs as <text>, and mark the cover page as <cover>. Then, write a style sheet that lists rules that titles should be bold, that text has this much space between paragraphs, and that the cover page's margins are extra-wide. Then, the computer knows to apply the formatting described in the style sheet to your document.

The benefits to semantic markup are immense. First, your boss's boss is

going to tell you to change your titles to italics instead of bold as soon as she sees the document. In the semantic system, you change the definition of a title element in one place and you're done; in the literal markup system, you need to go through the entire document and change every title individually—and then repeat when your boss's boss decides that no, you were right, it does look better in bold.

And did you catch the title on page sixty-eight down at the bottom? With semantic markup, because you didn't change fifty points in the document, you don't have to worry about whether you are still consistent or not. More generally, it is by construction impossible to have inconsistent style with a semantic markup scheme, because you define each style exactly once. With literal markup, you need to be on guard for consistency all the time.

In the literal markup world, you wear two hats at the same time: author and typesetter. In the semantic world, you wear the hats one at a time. When working on content, you are not distracted by stylistic junk. Some would describe working only on content and putting off stylistic issues until the very end to be “no fun”, but it is certainly more efficient. Of course, you are welcome to play around with the style sheet between writing every other sentence if you so desire.

The document you are working on now is probably not the only document you are writing during your career. Once you have a visual style that you like, you can save those definitions of titles, text, and cover sheet to use on every future document. In the literal markup world, you have to redo the margins on every cover page every time, duplicating a few minutes' effort with every document.¹

Along a similar vein, your company has a standard letterhead, and may have a graphics department that would prefer all documents to have a consistent style. In semantic-land, your company can distribute a single style sheet and ask that everyone apply it (even if they think it looks ugly; there are always dissenters in this system). In literal markup-land, the graphics department sends out a list of fifty rules everyone must follow when setting up their cover page, wasting everyone's time and guaranteeing that half of the rules won't get followed.

Finally, it is increasingly important that your document be available as a

¹There are of course tricks, like cutting and pasting the cover sheet from past documents and hoping the formatting follows. Even when they work, you can see that they are still inefficient relative to applying a style sheet.

PDF, a web page, and in your company's legacy TPS format. The semantic system, done right, is output-independent. You send the same document to one program that marks up titles appropriate for the printed page, and another that marks it up for the web. If you had to do literal markup appropriate for both the web and paper, it will look terrible in one or the other, and you'll just wind up writing and maintaining two documents.

Semantic markup is hands-down the way to format a document. It doesn't take any more cognitive effort or ability to mark up your title with `\title{Intro}` or `<title>Intro</title>` than it does to mark it up to read as **Intro**, so semantic markup provides all of the above benefits over literal markup for basically no cost.

Semantic markup v WYSIWYG Too bad Word is written from the ground up as a program for literal markup. If you read the manual, you will see that there exists a style editor, which claims to allow semantic markup. It lets you define paragraph types and character types, like a title style or a text style.

So the first tip, should you be using Word, is to use the style editor. Avoid hard-coding any sort of formatting; instead, define a style and apply that style.

But it's not entirely that easy, because Word will try its hardest to frustrate you. Word thinks it is smarter than you, so it will often guess the style you mean to be applying to a line, and sometimes revert styles back to where they were. Each element can have only one paragraph and one character style, but there are often reasons to apply multiple styles at once (blockquote on the cover page, italics in a title). Be careful to note where your styles are being saved. If they are being saved to your `normal.dot` template, then when you send your document to your colleagues, your styles won't go with.² Best of luck cutting and pasting between documents with different style sheets. There is a style organizer that allows you to move style sheet elements from one document to another; it is well-hidden (ask the paperclip for it) but it works. If you want one style for the web version of your document and one for print, you want too much.

The markup is always invisible: you need to click on the item while the style editor is up and then scroll through to see what is highlighted. This may

²Tip #2: write yourself a template. Depending on whether the Earth is in a Fire sign or a Water sign, you may need to send the template with your document.

seem trivial, but is frustrating if you have multiple, subtly different styles. And you will, because Word eagerly tries to manage the style list for you. If you italicize an item currently in the title style, then it will autogenerate a title-1 style. Now, when you change all of your titles to non-bold, the lone item in title-1 may or may not follow along.

In short, you *can* use Word for semantic markup, but only with discipline and patience. Word is a literal markup system, and the style editor is your window on Word’s internal means of organizing literal markup. It is not a full-blown semantic style sheet, as shown by its little failings above.

The bibliography If you are writing bibliographies by hand, you are wasting your life. Remembering where to put the commas, what to italicize, when to use the full first name and when to use initials, is the sort of work that a computer does easily and that we humans have trouble doing perfectly. Word’s demand that users need to hand-edit their bibliographies has no doubt cost the world literally millions of person-hours.

The correct way to do a bibliography is via a database. You provide one entry for each reference, typing out the author, the title, the publisher, et cetera. Then, the computer reads the database and puts the result in your document according to a style sheet such as that published by the University of Chicago or the APA. That is, the best means is via semantic mark-up: you tell the system that “Joe Guzman” is the author, and leave it to the computer to decide whether to print “Guzman, J”, “Guzman, Joe”, “J. Guzman” or what-have-you on the page.

OpenOffice.org wins points for including a bibliography editor. L^AT_EX includes bibtex. Word does not include one, although you can purchase one from a third party for a hundred dollars or so.³

2 Multiple views

A major contribution of the IT era has been to allow multiple views of the same work. The paradigm of the product has moved from the book, which is a fixed, indivisible object, to the song, which is merely a brief view on something for which hundreds of other views exist.

³Of course, if you use an add-on like this, you won’t be able to email your document to a colleague (or yourself at another computer) for editing unless the recipient has also paid out the hundred dollars for the same program.

Your favorite pop song was probably recorded onto 24 tracks, and then loaded onto a sound mixer such as Audacity, a program built around facilitating multiple views of the music. There's a visual representation of the sound and of course the noise the thing makes. The data is multilayered, and the user can view/hear the final work with some layers processed, muted, inverted, et cetera.

Users of the GIMP or Photoshop are familiar with the same process: there are all these layers, and each can be viewed differently, separated, processed. At some point, you set a fixed view and publish it as the final image, but with both the visual and audio systems, the base version holds much more information than the final view.

Databases have what is literally called a view, but even without them, users are encouraged to think in terms of the root data existing in the database and what's on the screen being a slice of it. The root data needs to be taken care of, but mangle the view all you want; it's disposable.

HTML is the markup language used by web pages; it is plain text but then viewed via a cute renderer like Internet Explorer or Firefox. On your hard drive, file browsers give you a dozen perspectives on the same pile of bits. Almost all information processing in all media takes the views-of-base-data form. But there are three hard-and-fast exceptions to the paradigm: spreadsheets, word processors, and presentation software. There is a picture of the page on the screen, and that's the document. There are few ways to view the work differently when you're working on it than when the final output will be printed or displayed on somebody else's screen. With due creativity, you can find the outline view or other marginal shifts, but for the most part, these systems work by vehemently insisting that there be only one view. All of the document's information is present in all versions.

And that is one more reason why Word is a terrible program: it constrains the user in the classical paradigm of "one work, one view". Can I distribute drastically different views of the same work to different people? They would just be two different works, that you'll have to maintain separately. When I present to the world congress, is there a clean version that I can use? Nah, just hit <F9> to blow up your working copy to full-screen size.

This is clearly what many people wanted, and most are happy with it. The two-bit philosophy questions of which is the true version of the work evaporate; the conceptual structure of a root object which is viewed in different ways flattens out; our documents are just like they were in the 70s, but backlit. But being stuck in the seventies means that there is a clear and

evident ceiling in efficacy, because the ideal view for working on a project matches the output view in only the most simple and lucky of circumstances.

HTML gives us some hope for the future here, because the actual work is done in a markup language, and then there are hundreds of possible views of that marked-up text—and people have no problem with the concept. One person can read the work on his telephone, and one can hear it on her text-to-speech reader, and both agree that they've read the same document.

A compounding failing of Word is that it won't let me insert the wealth of expletives that it inspires in me. With the one work = one view paradigm, there is nowhere for me to leave personal comments to myself that won't go into the public version.

Computer programmers have a trick known as 'commenting out'. Because the computer ignores anything marked as a comment, a coder can mark functioning lines of code as a comment to see how the program would run if that line were eliminated. It's a sort of purgatory for code that should maybe be deleted, but the judgement hasn't yet been handed down. Similarly, I write more than enough prose that is maybe a bit too verbose to put in the final work. I am reluctant to delete it, because I spent ten minutes composing that paragraph, but commenting it out is painless and reversible.

In Word, I am unable to leave personal notes to to guide myself, and I am unable to comment out sections that should probably be deleted. That is, Word gives me fewer tools to write with so that it can enforce its intuitive paradigm.

3 Intuition and lying to the user

A book entitled *Design of Everyday Things*, by Donald A Norman, very clearly had an influence on the design of many of Microsoft's products. It in turn was influenced by what was trendy at the time (1988): the original Macintosh features prominently, there is a whole page on the promise of hypertext, and he complains about EMACS. In his section on Two Modes of Computer Usage, he explains that there's a third-person mode wherein you give commands to the computer, and then the computer executes them; and there's a first-person mode where you do things your own darn self. Like telling the computer to multiply matrix A by matrix B versus entering numbers into the cells of a spreadsheet. At the ideal, you can't tell that you're using a computer; the intermediary dissolves away and it just feels like

working on a problem. Of course, some tasks are too hard for first-person execution, as Mr. Norman explains: “I find that I often need first-person systems for which there is a backup intermediary, ready to take over when asked, available for advice when needed.” This paragraph, I posit without a shred of proof, is the genesis of Clippy the Office Assistant.

Although Mr. Norman points out that we feel more human and less like computer users when we are in first-person mode, it is often a terribly inefficient way to work. A word-processor document is *not* like handwriting a letter, so pretending it is is sometimes folly. For example, you don’t hard-code numbers: instead of writing Chapter 3, you’d write Chapter `\ref{more_rambling}` (LaTeX form; Word has a similar thing), and let the computer work out what number goes with the `more_rambling` reference.⁴

In the context above, first-person mode matches literal markup. Don’t write a note to the computer that it should find all titles and boldface them; instead, go and boldface them all the way you would if you had a highlighter and paper in hand. Third-person commands are inhuman, unintuitive, and how we get computers to make our lives easier and more efficient.

Forcing the user DOET has much to say about saving the user from him, her, or itself. Make it impossible to make errors, he advises designers. His shining example of good design are car doors that can only be locked from the outside using the key. There’s a trade-off of some inconvenience, but it is absolutely impossible to lock the car keys inside. Word clearly fails on this one: you want to hard-code your references? Feel free; in fact, we’ll make it hard for you to do otherwise, since doing otherwise doesn’t follow the metaphor of simply writing on paper.

More generally, a good design has restrictions: if you can only put your hand in one place on the door’s surface, then that’s where you’ll put your hand, and the door will open on the first try. What about L^AT_EX? It gives you a blank page. You can type a basically infinite range of possibilities. This is where DOET leaves the command line: it isn’t restrictive enough to guide the user, and therefore is a bad design.

I think he’s got the interpretation entirely wrong: there is only one thing

⁴I used the LaTeX markup for Chapter `\ref{more_rambling}` here because it saves me the trouble of having to explain the seven-step process it takes to do the same thing in Word. And by the way, if you change the chapter’s title, all of the references will break and you’ll have to repeat the seven-step process for each reference.

that you can do with the blank slate that you get in EMACS, L^AT_EX, or a command line: read the manual (RTFM). Just as your car won't let you lock yourself out, you can't write a crappy document in L^AT_EX until you've gotten a copy of the manual and at least had half a chance to expose yourself to the correct way to do things. Mr. Norman again: "Alas, even the best manuals cannot be counted on; many users do not read them. Obviously it is wrong to expect to operate complex devices without instruction of some sort, but the designers of complex devices have to deal with human nature as it is." True, people won't read manuals unless you force them to. So force them to.

Metaphor shear Another problem is what Neal Stephenson calls *metaphor shear*. That's when you're happily working with a mental model in the back of your mind, and one day your metaphor breaks. Back to DOET: "Three different aspects of mental models must be distinguished: the *design model*, the *user's model*, and the *system image* [...]. The design model is the conceptualization that the designer had in mind. The user's model is what the user develops to explain the operation of the system. Ideally, the user's model and the design model are equivalent. However, the user and designer communicate only through the system itself: its physical appearance, its operation, the way it responds, and the manuals and instructions that accompany it. Thus, the *system image* is critical; the designer must ensure that everything about the product is consistent with and exemplifies the operation of the proper conceptual model."

This is where DOET overestimates computing. It's a book that's mostly about doors and faucets and other everyday objects. He's right that if you have to RTFM to work a door (even if the manual just says 'Push'), the door's design is broken. He's right that for complex systems, like panels of airline instruments, they should not work *against* intuition (e.g., if two levers do different things, they should look different). But he combines them into a false conclusion: complex systems should work with intuition so well that you shouldn't have to read the manual.

First, this is absurd in any setting but desktop computers. Would you feel OK if your pilot told you the plane was so intuitive that she didn't bother learning how to use it before the flight?

But back to the main point, making a word processor which is so intuitive to the user that he or she doesn't have to RTFM is a *much* more complex

task than making a manual-less faucet. If we needed to build a faucet such that it runs if the user presses it with his hand, bangs it with a pot, or bumps it with his elbow, that would be easy—put a button on the top. But to program a picture of a faucet such that the user can click on the thing, or double-click on the thing, or type R and all make the picture of a faucet run requires programming a call to the Run method for three separate events. If the user comes up with something that the programmer didn't think of, like holding down the alt key and clicking on the picture, then the user's metaphor shears. What your momma told you is true: it's easier to just present the truth than to weave a whole world around a lie.⁵

Mr. Norman's call for simple interfaces (he doesn't really say anything about metaphors to physical objects, but he does talk about simple mental models, and for most of us that means physical metaphors) therefore leads us down a supremely difficult path: first, the program designer must lie to the user by presenting a metaphor that is easy for the user to immediately guess at. Then, the designer must now design the program so that anything the user does, no matter how unpredictable, will cause the program to behave in the correct metaphorical manner. This is a very high bar, to the point that a program as complex as Word simply can not achieve it.

Ease of initial use The benefit of the intuitive interface is that you don't have to read the manual.⁶ You can jump in and go. Aunt Myrtle only writes one letter a month, so making her spend an hour reading the introduction manual—which she will entirely forget by next month—is inefficient and bad design.

But ease of initial use is only important for those items that we only use once or occasionally. Think of the things you use every day: your preferred means of transport may be an automobile, a bicycle, or your shoelaces. You spend all day typing with a QWERTY keyboard. Perhaps you play a musical

⁵For those down with the lingo: every event has to have a method for every object, which is dozens of events times dozens of objects equals hundreds of things that could go wrong with the metaphor—assuming you got good rules about passing the right events to the right objects to begin with. Inheritance doesn't help because most of the time the inherited methods don't quite work as they should, leaving you with objects which almost fit the metaphor.

⁶By the way, I rarely find intuitive interfaces to *actually* be intuitive. They're designed around certain target users whom I'm evidently incapable of thinking like. More generally, the concept of having an intuitive interface assumes that the intuition of everybody on Earth is exactly the same.

instrument. The fact that you are reading this indicates that you are literate. None of these things are intuitive. You spent time (in some cases, years) learning how to do them, and now that you did, you enjoy driving, riding, playing, and reading without thinking about the time you spent practicing.

Simply put, not having to read the manual is massively overrated. If a person is going to use a device for several hours every day for the next year or even the next decade, then for them to spend an hour, and maybe even weeks, learning to use the device efficiently makes complete sense.

Feature creep Mr. Norman is right that we shouldn't want to have to RTFM for simple, everyday tasks. Writing a letter or one-page paper is so common that his principle that it should be manual-less should probably apply. Further, we have the technology. However, as I've learned ever-so-painfully, writing a book is an order of magnitude more technically difficult. Programs like Word and Scientific Word imply that writing a letter and a book are identical, just a matter of extent, when in the end they aren't: one has a valid paper metaphor attached, which programmers can easily implement, and one does not. A good word processor, then, would let you do basic things without effort, and then put its foot down at some point. You get all the tools you need to write a business letter, and then if you want more, you'll need to get a new tool with a manual. Clearly, nobody is ever going to write a program like this. To some extent, this is a good thing, since it pushes technology forward, but at the expense of annoying users who have to sit through half-appropriate metaphors badly implemented. Mr. Norman writes about creeping featurism as an evil which pervades all of design, and he's right: nobody ever says 'I'm done.'⁷

The right way to implement this would be a simple graphical front end to the basic features of a less metaphor-laden back-end program. When you've sapped the offerings of the graphical front end, you'll have a bearing when you RTFM on the less intuitive stuff. This is how a host of Unixy programs work, but the front ends also eventually succumb to featurism. Scientific Word takes it to the extreme, by trying to give you a button for every last feature and refusing to admit that it is a front-end—perhaps because it is an

⁷There is a stand-out exception to this: \TeX was done in 1988, after nobody claimed the author's cash prize for finding bugs, and the code base has not changed since then. The add-on, \LaTeX , was cemented in 1994. Authors who want to change something in the system must add a package to the base systems.

expensive front-end to free software.

Since no programmer will ever have the discipline to admit that their manual-less tool will work only for a limited range of tasks, the discipline falls upon the user to realize that it's OK to use simplifying metaphors for simple situations, but complex tasks require tools that don't lie to you.

Word is carefully built from the ground up to be intuitive, not to be efficient—and it lies to you every step of the way to give the impression that the system actually works the way you intuitively guess it does. The next section describes how even the smallest intuitive but inefficient detail can add up to immense time costs in a system you use all day, every day.

4 Ergonomics

Google recently put out an RSS reader. It's pretty cute, and I personally have switched to it.

If you aren't familiar with RSS, then that is no matter here (it's a syndication system for web sites). The interesting feature of the reader for our purposes is that the j key will let you go down in the list of headlines. Yes, j, as in *jo down*. K, as in *kup* goes up in the list. No, there is absolutely nothing mnemonic about the J and K keys, but they *feel wonderful*. I assume you knows how to type properly, with hands on the home keys; I generally find my hands are on the home keys even when I'm just staring at the screen, and my hand doesn't need any help from my brain to find the little nubbin on the J key.

But that J key. It's the index finger of 90% of the world's dominant hand, and the keyboard is designed so that that index finger knows exactly where to rest. Moving down on the page is the most common operation, both in reading and even editing, so it makes complete ergonomic sense to attach this to the strongest finger of the strongest hand. Even the lefties will have no problem with it.

But it flies in the face of all mnemonics. Maybe you can come up with some word having to do with the process of scrolling down that begins with the letter J, but I've got nothin'. Nor could I think of a more efficient keymap.

I personally think the use of the J key is easy to learn because of its ergonomic delight. But it throws ease of initial use out the window. You want to use the nifty hotkeys? Then pretend you're literate and read the instructions.

You use your word processor or RSS reader for the first time once, but may subsequently use it five days a week every week for the rest of your life. Do the math: an extra minute a day lost due to an inefficient-but-intuitive interface means about four and a half hours a year lost. An interface which works *against* intuition can be destructive, so if U went down and D went up, we'd have to write off the application as hopeless, but J doesn't work against anything. It's just a gesture.

Within a week of Google's RSS rollout, Bloglines, a competing RSS aggregation service, added a little header to its page: "You can now navigate through Bloglines with hotkeys[...]: j - next article k - previous article [...]"

Anybody familiar with the OpenOffice.org internals? bdammm (at) openoffice (dot) org will give you a hundred bucks to write code to have J move the cursor down a line (plus a handful of other keystrokes like K).

The war Lest you think this J thing is some sort of recent meme, it all comes from vi, a text editor written in 1976. I am using a version of vi (vim) to write this right now. Let's pause for a second and let that sink in: most programs have a shelf life of about six months, and this guy wrote a program thirty years ago which is still in somewhat common use today. j goes down, k goes up, {jfw will go to the first instance of the letter w in your paragraph. Also, since I can't stand seeing that unclosed open-bracket, I have to tell you that }j%d% will delete a parenthetical remark in the first line of the next paragraph. Which is all to show you that Mr. Joy, the author of vi, fell soundly on the efficiency side of the efficiency vs intuition scale—and that is why his text editor has survived for thirty years, and is being imitated by cutting-edge web services.

We sometimes like to write documents which actually have Js in them, and vi thus has modes: in editing mode, j goes down and d\$ will delete the rest of the line; in insert mode, the j key puts a j on the screen, and typing d\$ puts gibberish on the screen which quickly reminds you you're in the wrong mode.

There are two competitors to J. The first is the ctrl-D school, rooted in EMACS, written by a certain Mr. RM Stallman. EMACS's keymap is sort of like vi's, in that it's not particularly intuitive, but once you've learned it, you're done. However, it's a compromise along the efficiency vs intuition scale, because you don't need to deal with modes (more intuitive) but reaching for the ctrl key all the time is not nearly as pleasant as twitching

your index finger to hit the j key.⁸ The EMACS vs vi war is a long-standing one, which is just silly, because they're of basically comparable efficiency. No, the real drain on the economy is the other school—the down-arrow school.

Let me take a paragraph or two to make this as clear as possible: the down-arrow school is a total failure when it comes to efficiency. On my screen right now, getting to the first w in the last paragraph via arrow keys is 27 keystrokes (using ctrl-arrow to go by word where possible). It's about three or four seconds for a single navigation. Do forty three-second navigations in a day and you're already up to nine hours in a work-year—a full work day a year just hitting the arrow key. You get to multiply by your wage to see what your company is spending per annum to facilitate ease of initial use. Even if it's one tap of the arrow key, your hands are already off the home keys; going off and on again is another half-second. If you do a hundred arrow-key navigations in a day (and if you're an office worker who does a lot of writing, you probably do closer to a thousand), that's another full work day a year just moving your right hand back and forth between the arrow keys and the home keys.

There is only one school that fails with such vehemence that it makes the down-arrow school look like Nirvana: the mouse school. In the mouse school, you take one hand—typically your dominant hand—off of the keyboard entirely, reaching to some part of the desk that is ergonomically suboptimal (because your keyboard is already in the optimal location). You position your hand on the mouse, and then move the cursor along the screen. It is an analog device, so aim and precision matter, meaning that some people simply do not have the eyesight and dexterity to use the mouse at all: try getting Aunt Myrtle to highlight the letter *i* in a font where that letter is one pixel wide. You guide the mouse to the pixels that are by the word you want to change, click, and return your hand to the keyboard. The entire process can easily take more than four or five seconds, just to position the cursor. And if you have to scroll through the document to find the point, that's easily ten seconds as prelude to a single edit.

The rabidness of the aforementioned text editor wars comes from the fact that text editing absorbs a huge amount of one's life. If you're like most office drones, most of your time at the computer is spent writing and editing plain text—and you're just one office drone; there are millions in the U.S.A. who are all operating computers basically identical to yours, using a down-arrow

⁸The joke is that EMACS stands for escape meta alt control shift.

school text editor of some sort. Sure, there are people doing flashy data-slinging with big servers, but the bulk of computing is the literally billions of person hours per year spent editing text. Now multiply that half-second to move the right hand to the arrow key; at this scale, it adds up to literally millions of person-days per year spent on making that little twitch. With an entirely straight face, I can say that on the order of a billion dollars per year is spent on paying people to hit arrow keys.

When the programmer guys got together and wrote whatever it is you use to write your documents and navigate your web pages, they had all of the paradigms at hand. Half of these guys are using EMACS or vi themselves. We get frustrated when we ask Mr. Computer Geek for help and he (always a boy, eh) comes back with over-everyone's-head exposition about just opening up `regedt` and doing a quick `ctrl-f` for `HKEY {343-f2ea53e}`. Less blatant but just as insidious is when Mr. Geek assumes you are an idiot. He knows that he knows more about PCs than you do, therefore you are dumb and wholly incapable of learning the reams of knowledge that he has compiled. I have been at many a workplace with IT departments that are stocked with such people; it's only some vestige of courtesy that keeps them from installing drool-guards on all the company keyboards.

Of course, the IT department is thinking about the worst-case users. But when was it ever efficient to force everybody in a several-hundred person organization to work with exactly the tools that the least-able could work with? You may have a legally blind worker at your workplace, but that doesn't mean that every computer in the building needs to operate exclusively at super-magnified resolution. A reasonable approach would be a system where you could select between the various schools of navigation. Most versions of vi let you do this (and EMACS allows `ctrl-D` and down-arrow), but few down-arrow school programs include a vi or EMACS keymap as well.

And so, after dealing with such people for too long, I take Google's j and k keys as a slight victory in a long battle against the forces of condescension. It's just two keys, a far cry from a word processor with a full vi keymap, but it's a sign that the guys who designed and programmed the system felt that it was more important to make usage efficient than to make it drool-proof. As such, it gives me hope that maybe the software of the future might focus on long-term efficiency over the quick sell.

Formatting and ergonomics Beyond editing, all this applies to formatting in Word too, because you have to use the mouse or an absurd amount of tabbing and arrowing to navigate the menus and dialog boxes to get to the option you want to change. For almost every step of the way, Word eagerly picks intuition over efficiency.

Of course, the most commonly-used features, like boldface, have their own ctrl-key combination, to at least save the user mouse and arrow-key inefficiency for the dozen most commonly-used operations. Also, you can use alt-F to access the File menu, alt-E to use the Edit menu, et cetera.

But even having the few control-key combinations you do have creates problems, because there are only 26 control-letters to use. If they are taken up with the typesetting features of Word, then they can't be used for the plain old editing of text. EMACS and vi give the user fifty-odd keystrokes that edit text (I'm guessing because I couldn't possibly count them all); Word gives you cut, paste, copy, and that's about it. For every other editing task, you have to make do with the arrow keys. Since the majority of your time putting together a paper is spent writing and editing, having so many keystrokes at your fingertips for formatting but almost none for editing is backward.

There is no place in Word's intuitive editing model for a key combination to delete a word at a time, or a key to repeat the last edit, or a key to jump to wherever you were last working. But such keystrokes provide *immense* speed gains to users who have taken the time to learn them.

One reason we have so many formatting commands is—once again—the lack of style sheets, which means that formatting is not produced by listing what you want the formatting to look like, but by applying it over and over again, which means that keystrokes to apply formatting are competing with editing keystrokes for frequency of use. It would be nice to have a dedicated editing program plus a separate dedicated formatting program, but Word's DOC format precludes this.

5 The DOC format and standards compliance

The World Wide Web consortium (the W3C) maintains the standards for what is a valid web page, and they provide a validator for web authors to

use to check the validity of our own pages, at <http://validator.w3.org>.

Most authors could care less about validation. They figure that if it looks OK on the browser they're using, and maybe one other, then they're done. For example, try validating the home pages of the World Bank (265 errors) or the Brookings Institution (131 errors).⁹

Even as esteemed an organization as the Library of Congress (whose front page validates perfectly) has considered building web pages that violate standards to the point of only working in one brand of browser, but at least they were polite enough to float the possibility with a request for comments first. Tim Berners-Lee, the author of the original HTML standard and frequently credited as the founder of the Internet, wrote a comment that explained the importance of standards:

At the outset, we would like to stress that nothing in this letter should be construed as a criticism of Microsoft's Internet Explorer [...]. We would write the same letter if the choice was to offer support solely for Mozilla Firefox, Safari, or any other product. [...]

While a large proportion of the marketplace uses the Microsoft Internet Explorer to browse the Web, certain classes of users will find it either impossible or extremely inconvenient to do so. [...] Users with disabilities often must augment their browsing software with special assistive software and/or hardware ("assistive technology"). [...] In addition, some individuals with disabilities rely on alternative browsers (for instance, "talking browsers") that are designed to meet their specific needs. Users with disabilities rely on a standards-based Web to ensure that services they access on the Web will be usable through the variety of mainstream software and specialized assistive technologies that they use.

He also points out that when a security flaw is found in a product, people or institutions will often switch to a competitor until the security flaw is patched. That is, even we of decent eyesight would do well to keep a variety of readers on our hard drives (I use three). This is obviously only possible if a variety of readers can all understand the same document format.

⁹These stats are from validation attempts in late 2005. I sincerely hope they do better if you try them today.

5.1 Extending the standards

So standards are good. But despite the obviousness of that statement, folks still insist on not complying.

Surely, the most common reason for ignoring a standard is that it does not allow for some form of expression that the author eagerly wants to use. But the author needs to bear in mind that freer expression bears all the costs of broken standards. My favorite Thai restaurant, *Thaiphoon*,¹⁰ has a website that I sometimes check so I can order ahead. When I open it in Firefox, I get a notice that I need to get the Flash plugin to view the site. Since I'm checking from a heavily restricted work computer, I can't install Flash, and often wind up eating at the Chinese place instead. But what happens when I visit the website in a Flash-enabled browser? I get a menu. A plain English text menu, swooshing in from one side.

Or consider the sad state of email. Like a restaurant menu, about 100% of email is also plain text. You tell people things, using words. For about 40 years, there has been a standard (ASCII) that allows different programs to interpret text correctly. Ah, what Nirvana: all the information we need to get across can be gotten across with an easy and supremely well-supported standard. If the UN worked this well, we would have world peace. In fact, now that the computing world is increasingly international, there are more character sets than English-centric ASCII, but nearly every known language is supported by the Unicode standard (yes, Ogham, Ugaritic, Deseret, and Limbu are in there.) Yet people increasingly throw the standard out and encode the text into a word processor document in a proprietary format. If you're lucky, you have a word processor that can read the proprietary-format documents your colleague emailed. For example, if the sender has Word 2000 and the recipient has Word 95, communication won't happen. Putting plain text in a word processor document—even with a bit of extra formatting—is exactly on par with putting a plain old menu in a Flash plugin: yeah, there's a little more glitz, but it comes at the price of potentially excluding, imposing work upon, or alienating the reader.

Of course, word processor documents are nice because they do provide extensions on top of plain text. They let you control the font and layout that the recipient sees in ways that plain text can only approximate. Flash certainly does things that HTML will never even think of supporting. But there is a trade-off that many people ignore, under the presumption that

¹⁰CT & S, NW DC. Try the Panang tofu.

everybody is just like them. ”Well, *I* have a copy of Word 2000 and an email client that displays web pages, so everybody else must too. My eyesight and dexterity with mouse and keyboard is fine, so my recipient’s must be too.” In a social context, the presumption that everybody is like you is the source of a great deal of impoliteness, offense, and general unhappiness, and we teach people from early childhood to understand that others are not like them and that they should maintain standards of decorum until they know that the other party is OK with breaking them. Sure, we can wear the risqué t-shirt to work and maybe make some people smile, but we know that such free expression carries a trade-off in the form of a risk of offending some. We should do the same when writing documents: stick to the basic standards unless we have a reason to do otherwise and we know that the recipient is OK with our new-fangled alternative.

There do exist valid reasons to ignore standards or set out to establish new ones; e.g., the correct response to a spoken “thank you” is “you’re welcome”, but it is accepted custom to send a “thank you” email but not a “you’re welcome” email, because that sort of thing just sort of clutters up the in box. But those who ignore the standards for no reason or for lousy reasons (“I don’t have to say thanks—he owed me.”) are just rude.

Bringing it back to the subject at hand, Word establishes its own standard, when it doesn’t have to. First, users often write a Word document when a simple plain text file will do. An email with no text in the body but a Word attachment with a single paragraph of plain text is a waste in every sense.

Second, there are standards that do approximately everything a Word document does, such as HTML. You can probably think of a few things that you can do in Word that you can’t do in HTML. You can also probably live your entire professional life not using them.

5.2 Alternative tools

Microsoft goes out of its way to make its DOC format opaque, because users are better locked-in if they can only edit their colleagues’ documents with Microsoft tools. But I promised you a paper that does not discuss Microsoft’s business strategy, but how Word’s design hurts your efficiency. The closed-format design means that, by definition, the only way to edit a Word document is in Word.

There are literally hundreds of editors for a \LaTeX or HTML document.

You can use anything that can read ASCII-formatted files—even including Word. That means that a market has sprung up that eagerly attempts to appease the needs and skills of different users. As above, EMACS and vi are specialized text editors and therefore have dozens of commands to just edit text, but there are literally hundreds of other text editors that I didn't mention; pick the one that most fits your lifestyle and run with it. For Word documents, you have no choice but to edit them in Word.

On the output end, there are a wide variety of programs that read L^AT_EX-formatted documents and display them via formats like HTML, PDF, or plain text. Because the file format is open, many people have implemented programs to process L^AT_EX-marked text to produce interesting new output.

Meanwhile, the only thing you can do with a Word document is open it in Word. If Word is not to your liking for any reason, you are stuck. If you need to output something besides Word DOC format, you had better hope that Word allows you to do the conversion.¹¹

XML The more tech-savvy readers know that the next version of Word uses the extensible markup language (XML), which is a commonly-accepted standard for semantic markup. However, this is slightly misleading. First, there is not yet a mechanism to write your own style sheets as I described above. Markup like `this` is valid XML, but it's just an elaborate way to say boldface. That is, Word takes a system designed for semantic markup and uses it for literal markup.

Second, the XML format depends on a document type definition (DTD) file, which is Microsoft-specific. Politics: although there exist open DTDs for text documents, including DocBook and OpenDoc, Microsoft is insisting on supporting one and only one XML schema: its own. It has applied for patents on that schema in the U.S. and Europe, and although it has stated that it will allow others to use its soon-to-be-patented technology for free, many are wary of whether the format will remain open.

¹¹Yes, many people try eagerly to write Word-document compatible extensions, with varying success. But the market for such extensions is absolutely miniscule compared to the market around plain text. OpenOffice.org will save DOC files as PDFs, by the way. Even if you are married to Word, you may want to download OpenOffice.org and keep it around exclusively as a PDF converter.

6 Alternatives to Word

Plain text Just open up the NotePad in Windows or TextEdit on the Mac and go to town—without formatting. You won't miss it. Need a graphic? Place a note like “[Place logo.gif here.]” Italicizing? Use underscores.

At the end of the project, you will want to have a beautifully-formatted document, which means moving your text to a document prep system or word processor, but put that off until the last minute. That is, spend the bulk of your weeks of editing and revising working on *content* and worry about format and visual appeal only as a final step. Because of Word's fundamentally first-person paradigm, you still need to change underscores to italics yourself, but (1) Word's macro feature can help with this, and (2) you may still save time and effort, because the editing features of text editors can add that much more efficiency.

Above, I had mentioned that it's nice to have a specialized program to do editing of content, and another specialized program to do formatting; the procedure here basically uses the text editor of your choice for the editing program and Word as the specialized formatting program.

HTML The Web has a text-based standard that can be successfully read by dozens of web browsers on all types of computer. HTML documents from the birth of the web in the mid-80s can still be read today. Even Word can read HTML.

HTML stands for HyperText Markup Language, and although the HyperText part is probably not too relevant to the discussion here, the Markup Language part indicates that this is exactly the sort of semantic language discussed above. This is especially true with the advent of Cascading Style Sheets (CSS). CSS lets you define a class, and describe how that class is to be formatted on the screen. Then, you mark up your text with class delimiters: this is a header, this is a digression. That is, HTML with CSS is exactly the sort of semantic markup language that we're looking for.

Your colleagues will be able to read these documents with their web browser, and even edit them with software on their computer.

L^AT_EX If you are in academia, use L^AT_EX. It was written for academic publishing, and universities are used to L^AT_EX users. It is designed around semantic markup of articles, books, and letters, and pegs them perfectly.

This document is written in it, and as you can see, it looks beautiful. Any journal you want your papers to be seen in accept (and frequently prefer) \LaTeX -formatted documents, and will provide you with a style sheet to apply to your document so that you can comply to their rules. Mathematics in Word looks amateurish, because only 0.02% of Word's buyers have equations in their papers; \LaTeX 's math typesetting makes you look smarter instantly.

It is not a strictly semantic markup, but a bit of a hybrid. I think it does a good job of combining the two, and if you want stricter semantics, then you are welcome to add `\defs` to the top of your documents to effect that.

One thing Word is good at, by the way, is deliberate inconsistency. If you want your first page in Helvetica, your second in Times, and your third page to be two-column format, this will be a pain in most semantically-oriented systems. But because Word's literal markup has no mechanism to impose consistency on the document, inconsistent formatting is much easier than in \LaTeX . So there's my token compliment to Word.

If you are not in academia, then you have a stronger compatibility-with-Word problem, but consider using \LaTeX anyway. Because there are reasonably effective (but imperfect) \LaTeX -to-HTML translators, you can think of the language as a document-oriented HTML-producing language, and can then send HTML to your trapped-in-Word colleagues. This method will especially benefit those who want to use bibtex or makeindex to autogenerate the end matter in larger works.

Now, the above methods require work and learning, but I hope by now you agree that spending time learning something that you will use every day for years is worth the effort. But, I'm not going to tell you how to go about learning HTML and CSS markup or which text editor to use. You know how to ask your favorite search engine for "efficient text editor", "HTML tutorial" or what have you. Many of these open standards and tools are entirely free, so there is at least no financial cost to downloading the tools and playing around. Better than the search engines is to ask your favorite guru for help; many are happy to take time to help a friend work more efficiently.

Also, because standard formats are so open, there is somebody who has fixed every problem you have, but it might be a separate tool. Some text editors include a spell checker, some depend on full-time external spell checkers. If you want to see the difference between your version of the document and the one your colleague edited, your editor may include a dedicated diff

mode, or you may need a copy of the `diff` program.

OpenOffice.org This is a word processor initially from Sun Microsystems. Its key claim to fame is that it can read and write Microsoft's DOC format very well, meaning that you can interoperate with your coworkers without their knowing that you aren't one of them.

Its stylist solves many of Word's style editor problems, so you may have better success with using it semantically. It has a built in bibliography database system. Maybe Mr. bdammm will get his wish for a basic vi keymap for efficient editing. The format is open, and so you can save to PDF. So complaints about some details are alleviated, but it tries to imitate Word to the point of imitating Word's paradigmatic failings. The literal markup, intuitive-over-efficient, and one work = one view paradigms remain.

7 Conclusion

A great many people have spent a great deal of time thinking about how to best edit and format text, and most of them have come up with solutions that look very, very different from Word. Part of the reason for this is that the authors of Word were writing for Aunt Myrtle, while the author of \LaTeX was writing a package for his own use; meaning that Word was built around ease of initial use, while \LaTeX was built around efficiency. There is no metaphor that one could make between an HTML document with a cascading style sheet and a physical paper with text—but this is liberating and allows for new possibilities and an easier time with formatting.

Perhaps you are stuck with Word, and company policy dictates that you write and maintain long, complex business documents using the same tool Aunt Myrtle uses to write her thank-you notes. Hopefully this paper has given you some ideas for working more efficiently: use the style sheet, stick to plain text where possible, maybe get a copy of OpenOffice.org on the sly for saving to PDF. But hopefully you have the liberty to take the effort and time to learn some of the other paradigms. It will take you days or even weeks, your first documents will look amateurish, and over the next several years of your career you will thank yourself over and over again as you gracefully produce output with truly efficient tools.