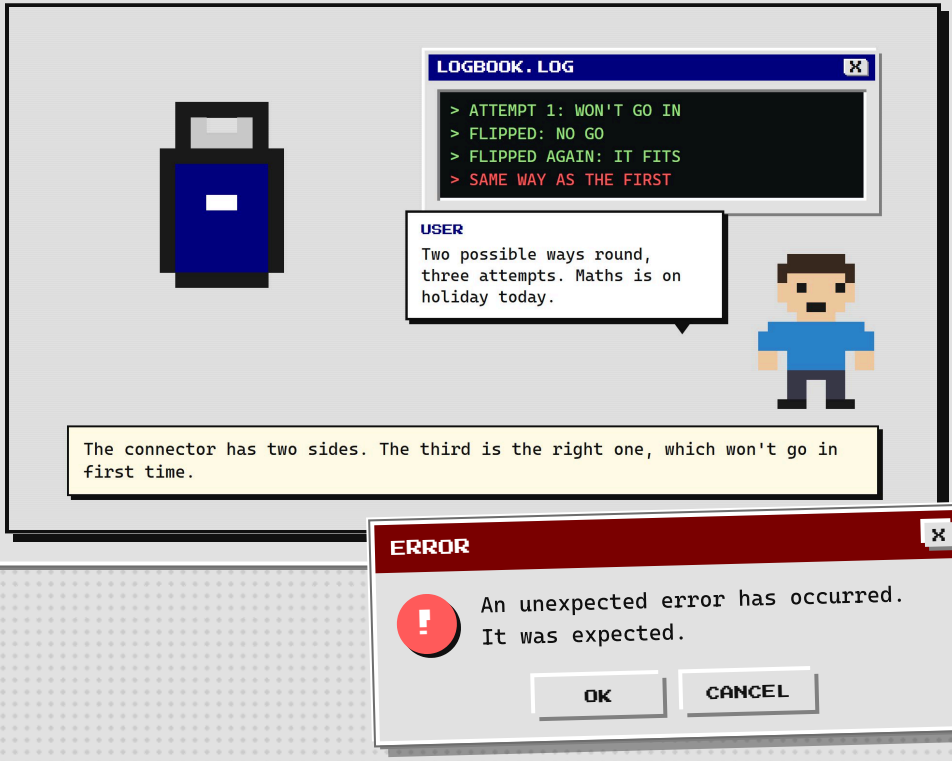
LEGGI_DI_MURPHY.EXE

MURPHY.EXE

Murphy's laws of computing, annotated by a sysadmin who has seen them at work



C:\MURPHY> read.exe /all /annotated
133 laws found. 0 errors. (so far)_

SILVANO DE POLI

133 LAWS + HOUSE COROLLARIES

MURPHY.EXE

Murphy's laws of computing, annotated by a sysadmin who has seen them at work

SILVANO DE POLI

To all the printers that taught me humility.
And to whoever made a backup last night: you know why.

CONTENTS

Preface 5
Instructions for use 6
1. The fundamental laws 7
2. Hardware: printers, cables and disks that listen 23
3. Helpdesk: users, passwords and other mysteries 45
4. Networks and the Internet: the invisible wire 63
5. Bug hunting 81
6. The art of writing code (and regretting it) 101
7. Projects, meetings and clients 121
8. Progress, updates and artificial intelligence 143
The house corollaries 167
Farewell 179
Colophon 180

PREFACE

My name is Silvano De Poli and my job is fixing things that broke all by themselves. I am a sysadmin and PC technician, I build websites, I work with artificial intelligence and I run the web agency RetroAI. In plain words: when something in the office stops working, the phone that rings is usually mine.

Over all these years I have learnt that failures are not random. They look random, but they have habits. They prefer Fridays. They love client demos. They hide when the technician arrives and come back as soon as the machine restarts. One day, facing a printer that had jammed on the very last sheet of letterhead, I realised I was not unlucky. I was inside a law.

That law, and many others, already existed. Edward Murphy, an engineer, gave his name to the most famous one: if anything can go wrong, it will. After him, programmers, economists, writers and researchers added their own observations, some very serious, others born as jokes and turned into proverbs. Computing has collected them all, with the same care it gives to the cables in the drawer: you never know.

This collection was born on my website, one law at a time. At first they were notes: the law, who stated it, a comment. Then the comments grew, themes arrived, and some laws even became small pixel-art videos. At some point there were a hundred and thirty-three of them, and I noticed that together they told a story: the affectionate, complicated relationship between people and machines.

There are three kinds of laws in this book. There are those **with a proper name**, tied to a real person: Moore, Brooks, Parkinson, Hofstadter and many more. For these I have tried to explain what they really say before poking fun at them. There are the **anonymous** ones, handed down from office to office like family recipes, which nobody signed because everybody lived them. And there are a few **corollaries of my own**, mostly about artificial intelligence, which I sign with some embarrassment and a lot of pride.

Every law has a comment. The tone is the website's: ironic, sometimes a little surreal, always affectionate. The most absurd anecdotes are, I assure you, made up. The most believable ones too, though I say so with less confidence. The technical explanations, however, are true, and I have tried to make them clear even to people who have never opened a computer, except to look for a screw.

At the end of each chapter there is *The sysadmin's advice*. It is the serious part, the one I would like you to keep in your pocket: practical, tested things that really help. They do not defeat Murphy, because Murphy cannot be defeated. But they make him work harder.

You can read the book in order or at random. You can keep it on your desk like a breviary and open it before an important update. You can quote it in meetings to sound wise, or give it to the colleague who keeps saying ‘it works on my machine’. This works too. On his machine.

One last warning. While I was putting these pages in order, the computer froze twice, the Wi-Fi vanished during an important search and the printer refused to cooperate with the test print. I am not saying this to complain. I attach it as a certificate of authenticity. The laws work. Enjoy the read, and save often.

INSTRUCTIONS FOR USE

User manual, model MURPHY.EXE. Read carefully before switching on.

- **Start-up.** The book can be opened at any page. No restart required.
- **Reading order.** Chapters go from the most universal to the most technical. Non-programmers can skip the fifth and sixth without permanent damage, but if they read them they will finally understand their programmer.
- **Dosage.** One law a day is recommended, preferably before touching a server. In case of overdose, involuntary quotations may occur during meetings.
- **Sysadmin's advice.** At the end of each chapter. They are the only serious parts of the book: to be applied, not just read.
- **Compatibility.** Works on screen, on paper and read aloud. Also tested in print, with mixed results.
- **Warranty.** The book does not prevent failures. It only makes them easier to recognise. In case of malfunction, try reading it again.

01

CHAPTER 1

CH 01 — FOUNDATIONS

The fundamental laws

Every religion has its sacred texts, and computing has these laws. Nobody carved them in stone: they were worked out in the field, one embarrassment at a time, by engineers, programmers, economists and people who just needed to print a form.

We start with the mother law, Murphy's, which was originally a piece of design advice and became our daily horoscope. Then come its close relatives: one adds the time of the disaster, one reminds us that the computer obeys to the letter, one claims that a program that runs first time is just taking a run-up.

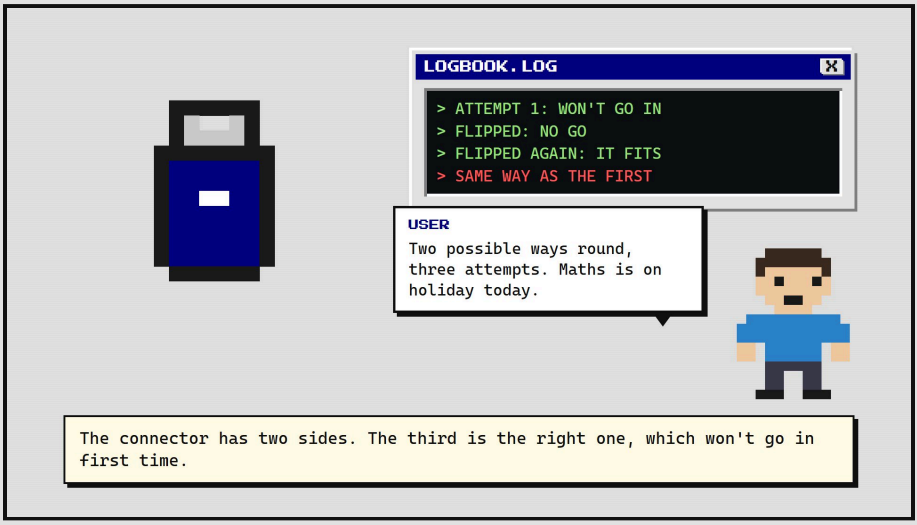
Along the way there are also laws born outside computing, in bureaucracy, economics, science fiction. We adopted them because they described us far too well: work that stretches to fill the time, the colleague promoted up to the wrong level, the eighty per cent of trouble that always comes from the same twenty per cent.

This is the chapter to read first and to quote in meetings. These laws work like Latin phrases: whoever says them instantly sounds wiser, even if the server is still down.

CONTENTS.DIR

01 Murphy's Law
02 Finagle's Law
03 Law of the literal computer
04 Law of the residual bug
05 Law of suspicious success
06 Hofstadter's Law
07 Parkinson's Law
08 The Peter Principle
09 The Pareto principle (80/20)
10 Hanlon's razor
11 Clarke's third law
12 Sturgeon's Law
13 Moore's Law

MURPHY'S LAW



If something can go wrong, sooner or later it will. This applies above all to connectors that fit both ways round.

— Edward A. Murphy Jr., *Edwards Air Force Base, project MX981, around 1949; exact wording and circumstances disputed*

It all started with an engineer, some sensors fitted the wrong way round and a ruined experiment. Originally it wasn't pessimism: it was design advice. If a part can be fitted wrongly, someone will do it, so design it so that it can't be.

We IT people took it as a prophecy. The USB cable has three sides: the right one, the wrong one and the right one that still won't go in on the first attempt. That's the quantum physics of ports for you.

I apply it with bureaucratic rigour. Before every update I fill in a form: 'Things that can go wrong'. The form runs to forty pages. The update freezes on page forty-one, which I hadn't foreseen.

FINAGLE'S LAW



Anything that can go wrong will go wrong, and it will choose the worst possible moment to do so.

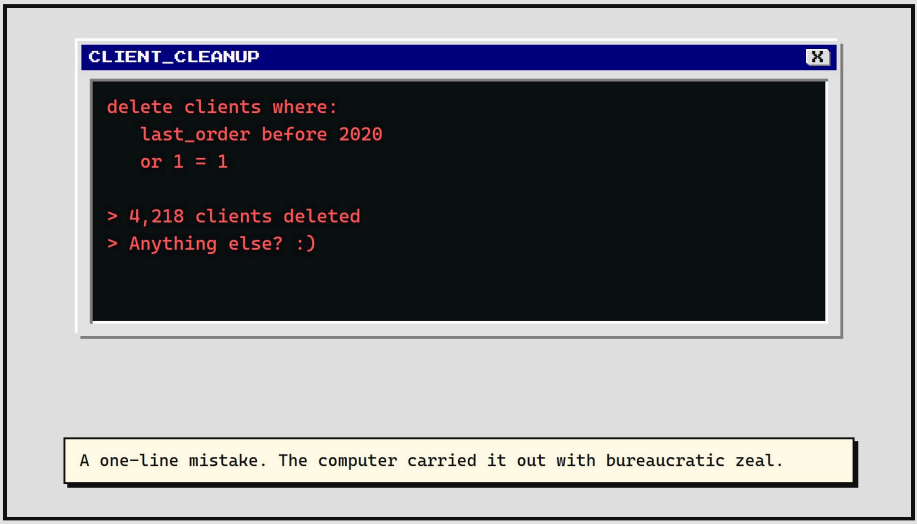
— Anonymous, *Variant of Murphy's Law that spread through American science fiction circles in the 1950s and '60s; uncertain attribution*

Murphy says disaster will come. Finagle adds the time: Friday at 17:58, while you're already switching off the monitor. It's the version with a synchronised calendar.

The server never breaks on a Tuesday morning, when you're rested and have your coffee. It breaks during the client demo, on year-end closing day, or when you're at your cousin's wedding with the laptop locked in the car. The fault has a diary, and yours is always open to it.

I tried to trick it: I pretended to go on holiday. The server didn't flinch. Then, the day I actually came back, the power supply caught fire. I don't know how it knew. I suspect the shared calendar.

LAW OF THE LITERAL COMPUTER



The computer does exactly what you told it to do, never what you wanted it to do.

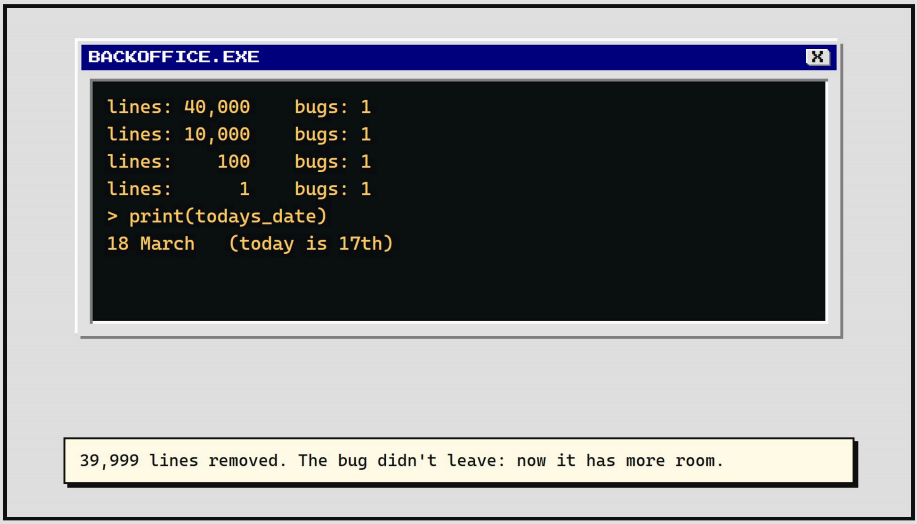
— Anonymous, *computing folklore*

The computer is the most obedient employee in history and, for precisely that reason, the most dangerous. It doesn't interpret, it doesn't intuit, it doesn't look you in the eye to work out what you meant. It executes. To the letter. With bureaucratic zeal.

Tell it to 'delete inactive customers' and, thanks to a misplaced comma, the inactive customers are all of them, and it deletes them all, with admirable efficiency. Then it asks if there's anything else. Ever so kind.

It's like having a genie of the lamp who studied law. Every wish has to be drafted with clauses, exceptions and footnotes. For non-programmers this is the true essence of the job: not explaining to the computer what to do, but anticipating every creative way it might misunderstand you. To be on the safe side, I now talk to the dog like this too.

LAW OF THE RESIDUAL BUG



Every program contains at least one bug, and every program can be shortened by one instruction. It follows, by induction, that every program can be reduced to a single instruction that doesn't work.

— Anonymous, *computing folklore*

It is the fundamental theorem of programming and, like every serious theorem, nobody has ever proved it, because everyone was busy fixing something else. The logic is flawless: remove one line at a time and the bug is always still there, clinging on like a limpet to a rock.

I tried to verify it in the lab. I took a forty-thousand-line management system and started removing lines. When I got down to one, the program printed the date. Wrong. By one day. Only in months with an R in them.

For non-programmers the moral is simple: perfect software exists, but only until somebody writes it. The moment you press the first key, the error takes a seat in the front row and starts taking notes.

LAW OF SUSPICIOUS SUCCESS

TERMINAL

```
> compile project
0 errors, 0 warnings
> run
all ok. time: 0.01 seconds

(too fast. too ok.)
```

The seasoned programmer doesn't cheer. He shuts the door and checks it all again.

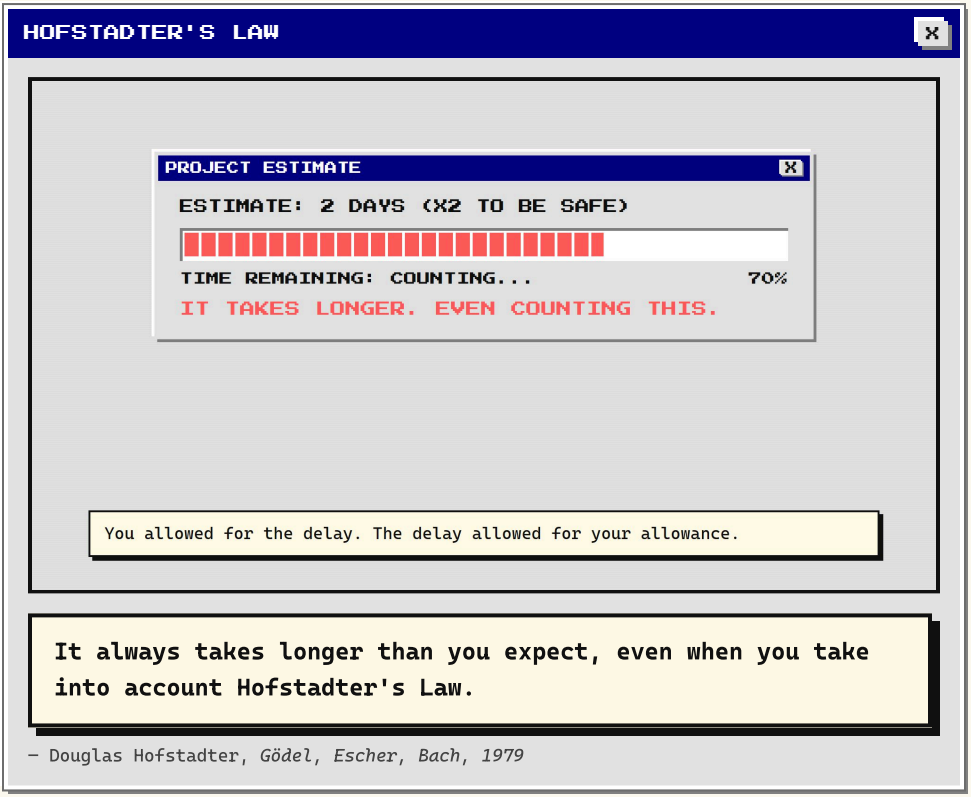
If a program works on the first attempt, it is not a good sign: it means the error is buried deeper and will show up at the worst possible moment.

— Anonymous, *computing folklore*

When an experienced programmer's code runs straight away without errors, they don't celebrate. They turn pale. They get up, close the door and check everything again, like a bomb disposal expert facing a parcel that ticks too little.

The reason is statistical. Software is a minefield where every step is wrong at least once. If nothing has gone off, it only means the mine is further ahead, and bigger, and waiting for the customer.

It really happened to me once: compiled, launched, perfect. I celebrated with a coffee from the vending machine. Three months later I discovered the program worked because it wasn't doing anything. It skipped the whole important part with great elegance. Since then, when something goes right first time, I treat it like an overly polite guest: with affection, and counting the cutlery.

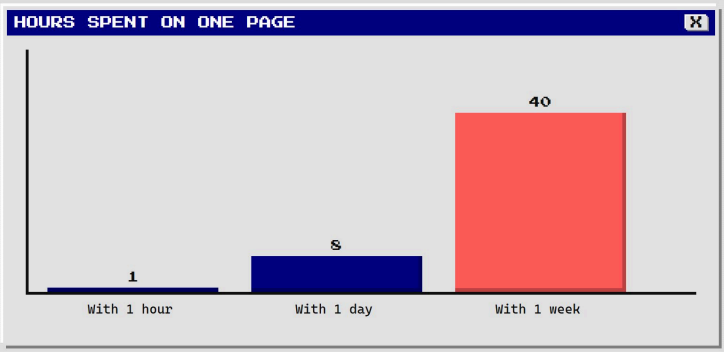


It's a recursive law, and therefore much loved by programmers. You estimate two days. You know you're always wrong, so you write four. But the law knows you know, and it takes eight.

It's the progress bar of life: 'Time remaining: 2 minutes.' Then 47 minutes. Then 3 seconds. Then 'Calculating time remaining', which is the elegant way of saying 'no idea'.

I asked an artificial intelligence to estimate my project's duration applying Hofstadter's Law. It replied: 'It will take longer than expected.' I asked how much longer. It's still processing. I gave it a ten-minute deadline, taking the law into account. I'll be back on Tuesday.

PARKINSON'S LAW



The page is identical. The one-week one had its font changed fourteen times.

Work expands so as to fill the time available for its completion. Computing adds a corollary: data expands to fill all the space available.

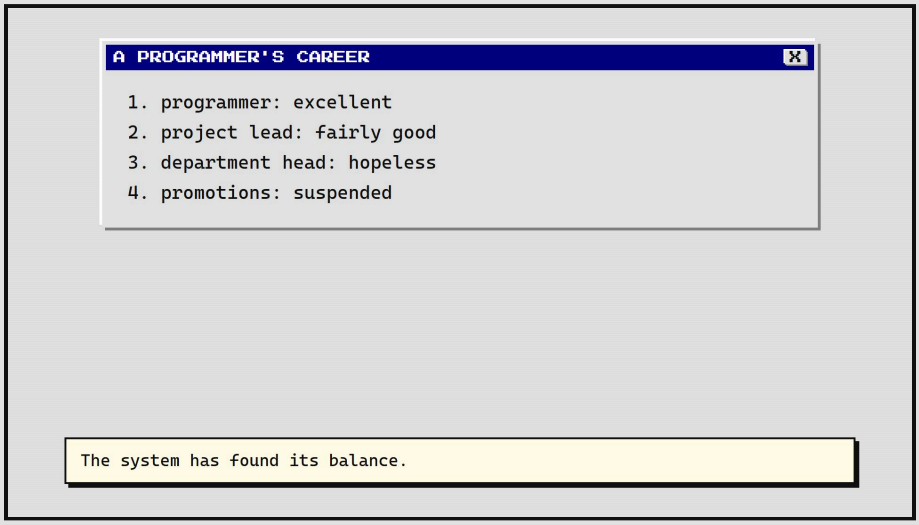
— Cyril Northcote Parkinson, *The Economist*, *November 1955 (data corollary: anonymous, uncertain attribution)*

If you have a week to write a page, it will take you a week. If you have an hour, it will take an hour. The page will be identical. The one-week version will just have had its font changed fourteen times.

Born as a satire on British bureaucracy, the principle has become the unofficial handbook of IT projects. Generous deadlines don't produce better software: they produce better meetings, meaning longer ones. And it applies to things as well as people. You buy a four-terabyte disk and within six months it's full, like a new garage.

At my office we tried to beat the law by setting fake deadlines, shorter than the real ones. It worked for a month. Then everyone started working out the real deadline, and now we have a spreadsheet that estimates it. We've been working on it for three weeks.

THE PETER PRINCIPLE



In a hierarchy every employee tends to be promoted until they reach their level of incompetence, and there they stay.

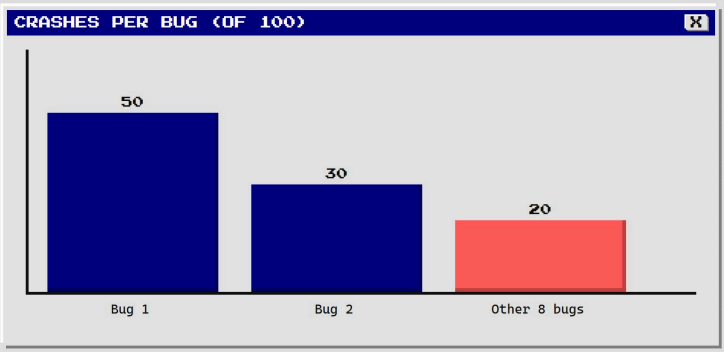
— Laurence J. Peter (with Raymond Hull), *'The Peter Principle', 1969*

Once upon a time there was a brilliant programmer. So brilliant that they promoted him to project lead. There he was less brilliant, but good enough to be promoted to head of department. And there, at last, he was completely incapable. So they stopped promoting him. The system had found its equilibrium.

The Peter Principle explains why, in many companies, whoever used to write the best code now spends their days making tables about tables. Being good at something doesn't mean being good at leading the people who do it. But promotion rewards the first skill by handing out a job that requires the second.

I avoided the problem brilliantly: I went self-employed. So I'm programmer, sysadmin, salesman and managing director all at once. I've reached my level of incompetence in four roles simultaneously. It's an optimisation.

THE PARETO PRINCIPLE (80/20)



Two bugs in ten cause eight crashes in ten. That twenty per cent works very hard.

In many phenomena about 80% of the effects come from 20% of the causes: for example, 80% of the problems come from a small fraction of the bugs or the customers.

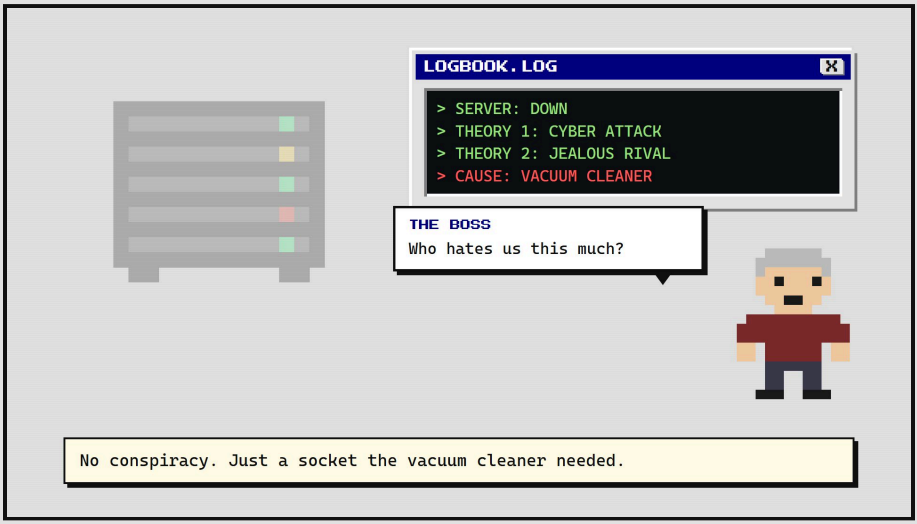
— Vilfredo Pareto, generalised by Joseph M. Juran, *Pareto's observation on the distribution of wealth (1896); name and generalisation by Juran (1940s–50s)*

An Italian economist noticed that a few people owned most of the land. Decades later a quality expert generalised it: a few factors cause most of the effects. Ever since, eighty and twenty have been the most famous couple in management consultancy.

In computing it works almost embarrassingly well. Eighty per cent of crashes come from a handful of bugs. Eighty per cent of support calls come from the same twenty per cent of users, and you know their names perfectly well. Eighty per cent of a program's value lies in a few features, which are precisely the ones hidden in the third submenu.

I applied Pareto to my desk. Twenty per cent of the cables cause eighty per cent of the tangles. The trouble is I don't know which twenty. I unplugged them all. Now it's tidy, but nothing switches on.

HANLON'S RAZOR



Never attribute to malice that which is adequately explained by carelessness or incompetence.

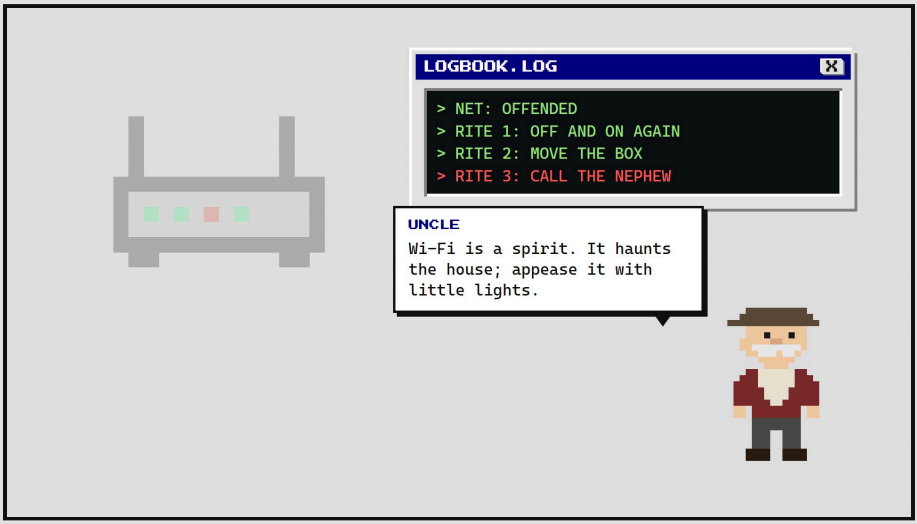
— Robert J. Hanlon, *Arthur Bloch, 'Murphy's Law Book Two', 1980 (similar ideas in much earlier texts)*

The server is down. First thought: cyber attack, foreign spies, a jealous competitor. Then it turns out the cleaner pulled the plug to use the hoover. No conspiracy. Just one free socket fewer.

Hanlon's razor is the support technician's best friend. When a user deletes the shared folder, they aren't sabotaging the company. They just clicked decisively on something they didn't understand. Which is worse, but at least it isn't personal. Keeping it in mind saves human relationships and blood pressure.

A tip: the razor works the other way round too. When you're the one who got it wrong, hope the others know it. I always quote it to clients before telling them what I've done. It works as a kind of diplomatic antivirus.

CLARKE'S THIRD LAW



Any sufficiently advanced technology is indistinguishable from magic.

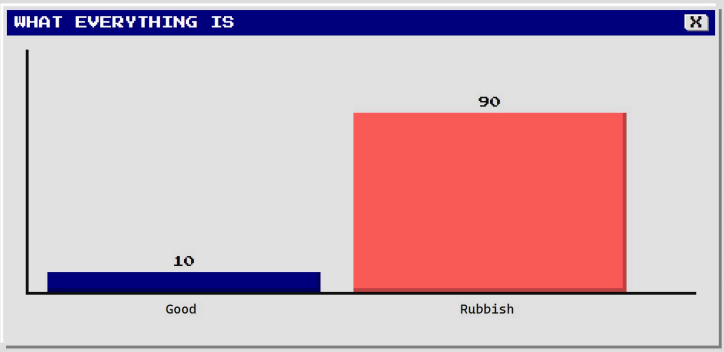
— Arthur C. Clarke, *'Profiles of the Future', revised 1973 edition (already in a 1968 letter to Science)*

A science fiction writer observed that a sufficiently advanced technology, to anyone who doesn't know how it works, looks like a spell. He had alien civilisations in mind. He hadn't seen my aunt in front of the Wi-Fi.

For her the network is an invisible fluid that inhabits the house and takes offence easily. It must be appeased by switching a box with little lights off and on again. If that's not enough, the box is moved. If that's still not enough, the nephew is summoned, recites incomprehensible formulas, and the box is healed. It isn't technology. It's liturgy.

The dark side of the law is that it also applies to those who sell technology: the more it looks like magic, the fewer questions people ask. I, on the other hand, always explain everything. Clients listen respectfully, nod, and then reboot anyway. Magic wins.

STURGEON'S LAW



True of science fiction, apps and plugins. Probably of this chart too.

Ninety per cent of everything is rubbish: it applies to science fiction, but also to books, films, programs and products of every kind.

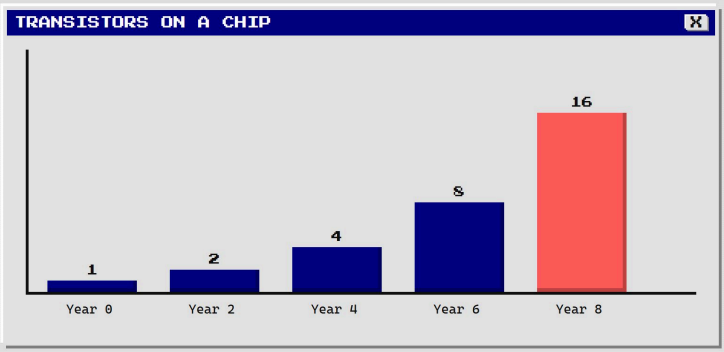
— Theodore Sturgeon, *Venture Science Fiction, March 1958 (also called 'Sturgeon's revelation')*

A science fiction writer kept hearing that ninety per cent of science fiction was rubbish. He replied: of course, because ninety per cent of everything is rubbish. End of discussion, applause, and a law that holds everywhere.

In computing it's practically a unit of measurement. Ninety per cent of the apps in the store are useless. Ninety per cent of the libraries on the Internet are unmaintained. Ninety per cent of the Joomla plugins I installed in 2011 are still there, and nobody remembers what they were for.

The beauty of the law is that it isn't pessimistic: it says the good ten per cent exists. The problem is finding it. I've built a career on it. My job, essentially, is sifting sand. Every now and then I find a nugget. More often, a driver.

MOORE'S LAW



In 1965 four dots on a graph were enough. It went on like that for half a century.

The number of transistors that can be crammed onto a chip doubles at regular intervals: at first every year, then, after an adjustment, roughly every two years.

— Gordon Moore, *Electronics, 1965 (forecast revised in 1975)*

In 1965 an engineer looked at four dots on a graph and said: it'll carry on like this. He was right for half a century. I look at four dots on the calendar and say I'll finish the website by Friday. Similar methods, different results.

In practice it means that every two years computers become twice as powerful. And every two years the program you use becomes twice as heavy, so the net result is that Word always takes six seconds to open. It's a cosmic balance. An eternal draw between hardware engineers and software engineers, and the referee is the Windows hourglass.

Today transistors are so small they're about to bump into atoms. My cousin says that at that point you'll just have to squeeze harder. He even has a special screwdriver for it.

THE SYSADMIN'S ADVICE

Murphy cannot be defeated, only outflanked. Before any serious job, be it an update, a server move or a change to the website, I follow four rules written on a real sheet of paper, stuck to the monitor.

- **A copy first.** A full, verified backup, made now and not ‘the other day’.
- **A way back.** If I don't know how to undo the change, I don't make it.
- **One thing at a time.** Changing three at once means not knowing which of the three broke the fourth.
- **A note for every step.** Date, what I did, what happened. In six months it will be the only reliable document.

You don't need expensive software: a notebook will do. A notebook doesn't update itself and doesn't ask for a licence. Murphy will strike anyway, but he will find the back door already open on his way out.

CHAPTER 2

CH 02 — HARDWARE

Hardware: printers, cables and disks that listen

Software is scary because you can't see it. Hardware is scary because you can see it perfectly well, and it looks back at you. The printer with the orange light, the cable that doesn't reach, the screw rolling under the motherboard with a sound very much like a laugh.

This chapter is about objects. The ones on the desk, under the desk and, in the worst cases, behind the desk, where the dust has developed its own microclimate. These are the most democratic laws in the book: you don't need to know how to program to have suffered them. You only need to have had to print a contract at five to nine.

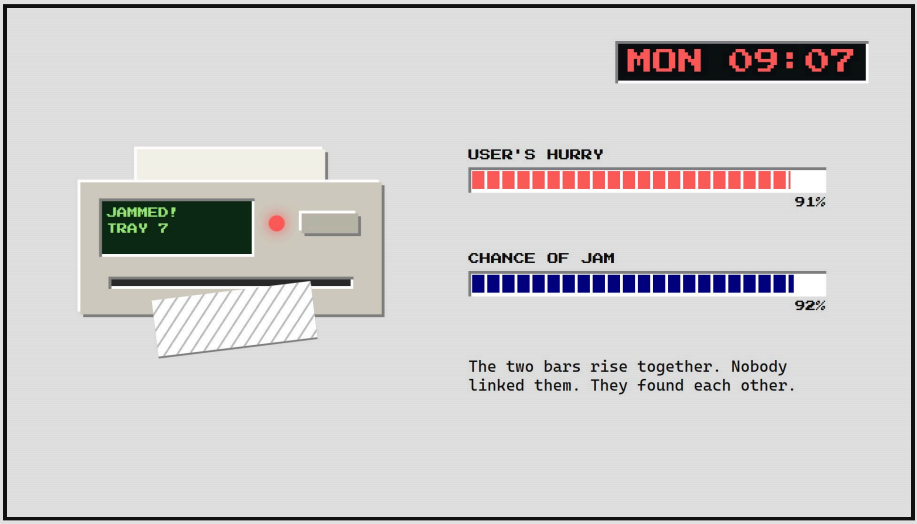
In the second part we go down into the disks, where data lives with its fears. The backup that isn't there, the one that is there but won't restore, the file that always lives on the other computer. Here the irony is subtler, because the pain is more real. A broken printer can be replaced. Wedding photos can't.

Read it with a screwdriver within reach. It won't help, but it's reassuring.

CONTENTS.DIR

01 Law of the impatient printer
02 Law of the clairvoyant toner
03 Law of the selective paper jam
04 Law of the phantom network printer
05 Law of the short cable
06 Law of the missing cable
07 Law of the quantum USB stick
08 Law of the runaway screw
09 Law of the expired warranty
10 Law of the optimistic battery
11 Law of the hostile projector
12 Law of the missing driver
13 Law of the backup you didn't make
14 Law of the psychic hard disk
15 Law of the untested restore
16 Law of the unsaved work
17 Law of the file on the other computer
18 Segal's Law
19 Kryder's Law

LAW OF THE IMPATIENT PRINTER



The probability that the printer will jam is directly proportional to how much of a hurry the person printing is in.

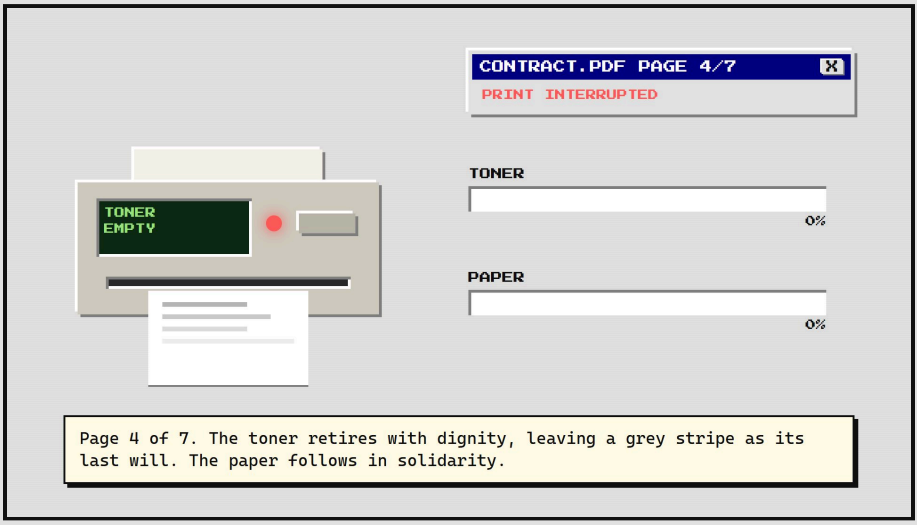
— Anonymous, *computing folklore*

The printer has no eyes, no ears and no connection to the clock on the wall. And yet it knows. On Monday at nine, with the train leaving at ten past, the printer senses the user's heartbeat the way a shark senses a drop of blood.

I once knew an office that for years printed only calmly, humming under its breath, so as not to be found out. It worked. Then one day an employee looked at the clock. Forty-two minutes of ‘warming up’.

A technician's advice: print important documents the night before. Not because the printer sleeps, but because the next morning you'll have a plan B, and it knows that.

LAW OF THE CLAIRVOYANT TONER



Toner and paper always run out halfway through the most important document of the week, never during the test prints.

— Anonymous, *computing folklore*

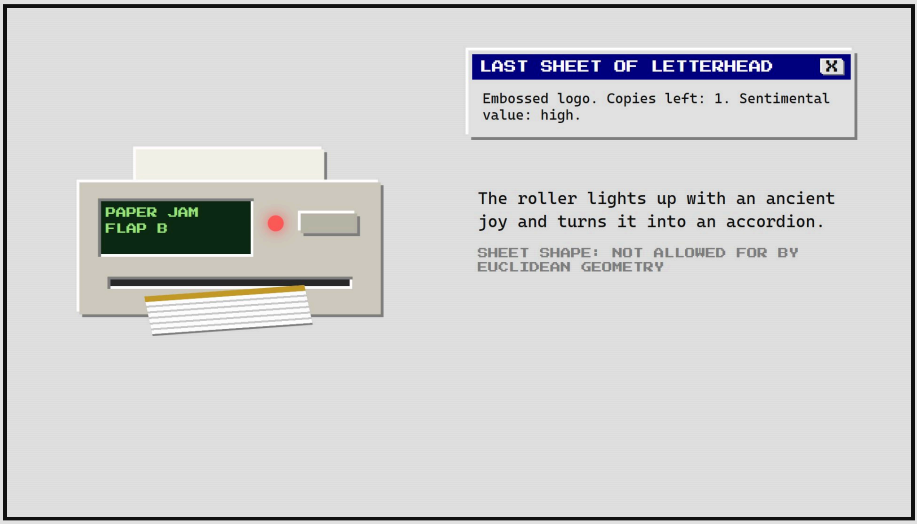
Toner is a prophetic substance. For three months it prints village fête flyers, recipes and photos of the cat without batting an eyelid. Then the contract to be signed arrives, and it retires with dignity on page four of seven, leaving a grey stripe as its last will and testament.

My cousin used to shake the cartridge every morning, out of habit, the way you shake a thermometer. He said it frightened the toner into staying. He had a drawer full of shaken cartridges. Not one of them ever lasted until a house completion.

Meanwhile, in the meeting room, someone asks whether you can print ‘just in black’. The answer is no: black is precisely the one that's run out.

Mini-tip: always keep a spare cartridge. It costs less than the embarrassment.

LAW OF THE SELECTIVE PAPER JAM



Paper only jams on the one sheet you can't reprint: the letterhead, the pre-printed form, the last good sheet.

— Anonymous, *computing folklore*

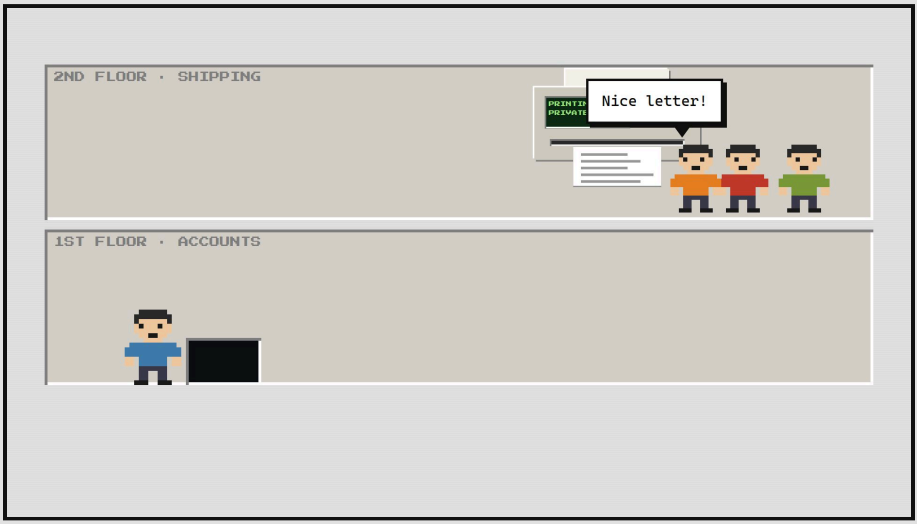
Printers have a sommelier's talent. Faced with a two-euro ream they are as meek as lambs. But as soon as you put in the last sheet of letterhead, the one with the embossed logo, the roller lights up with an ancient joy and turns it into an accordion.

I've seen colleagues open flaps I didn't know existed. Flap A, flap B, flap ‘do not open during operation’. Inside, the sheet was folded into a shape Euclidean geometry does not allow for.

Meanwhile the printer's display showed a little animated icon of a hand pulling out the paper. Taking its time. Like someone explaining to a child that the ball has gone into next door's garden.

Real advice: before using the precious sheet, do a test on ordinary paper. Always.

LAW OF THE PHANTOM NETWORK PRINTER



The confidential document you send to print always ends up on the printer on the other floor, in front of the wrong person.

— Anonymous, *computing folklore*

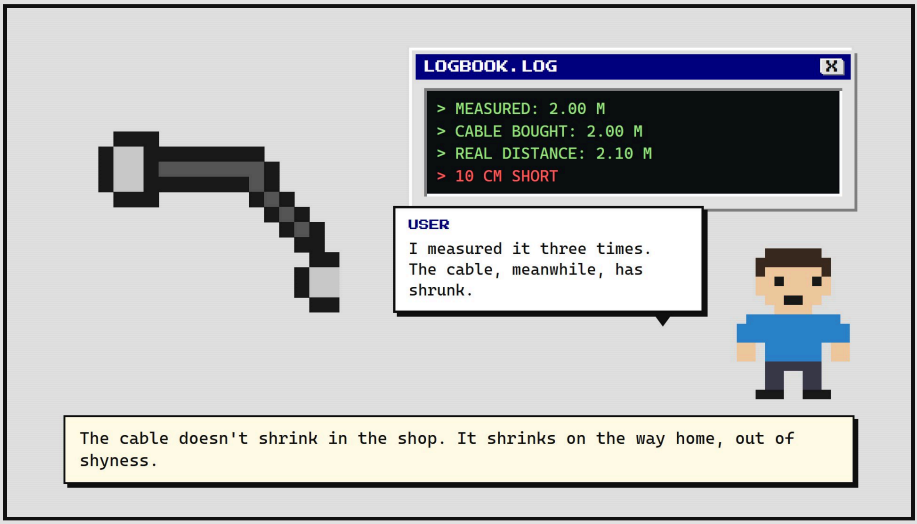
Every company has a printer that nobody installed. It appears in the list with a name like ‘HP_SECOND_FLOOR_OLD’. Nobody knows where it is. But it's always the default when you print your resignation letter.

Once I followed a document's journey through the whole building. It had set off from accounts, passed through the warehouse and been collected, read and discussed by the shipping department. Three people told me ‘nice letter’. It wasn't mine.

The phantom printer doesn't make mistakes. It chooses. It keeps a secret archive of everything it wasn't supposed to see, and one day it will write its memoirs.

A sysadmin's advice: check which printer is set as the default before pressing ‘Print’. Two seconds that save careers.

LAW OF THE SHORT CABLE



Every cable is always ten centimetres shorter than the distance it needs to cover.

— Anonymous, *computing folklore*

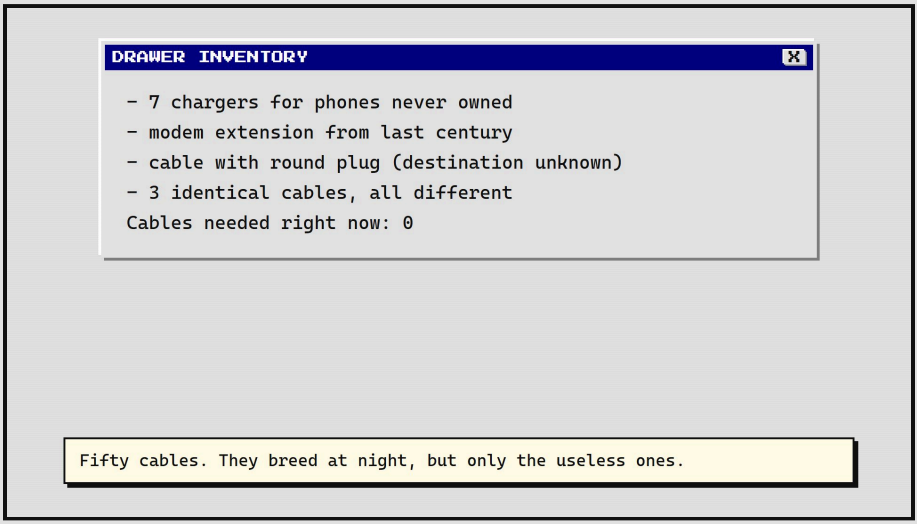
It's a universal constant, like the speed of light. If the socket is a metre away, the cable is ninety centimetres. If you buy the two-metre one, the desk moves. Nobody saw it move, but the tape measure doesn't lie.

A friend of mine tried to cheat the law by buying a ten-metre cable to connect a monitor half a metre away. The cable arrived. He also moved house, and the new socket was ten metres and ten centimetres away. Coincidence, say the sceptics.

The short cable is also a life coach: it teaches patience, humility and, above all, crawling under the desk with a torch in your mouth.

A technician's advice: measure, add half a metre, and buy that. A long cable can be coiled up; a short one can't be stretched.

LAW OF THE MISSING CABLE



In a drawer containing fifty different cables, the one you need right now is never there.

— Anonymous, *computing folklore*

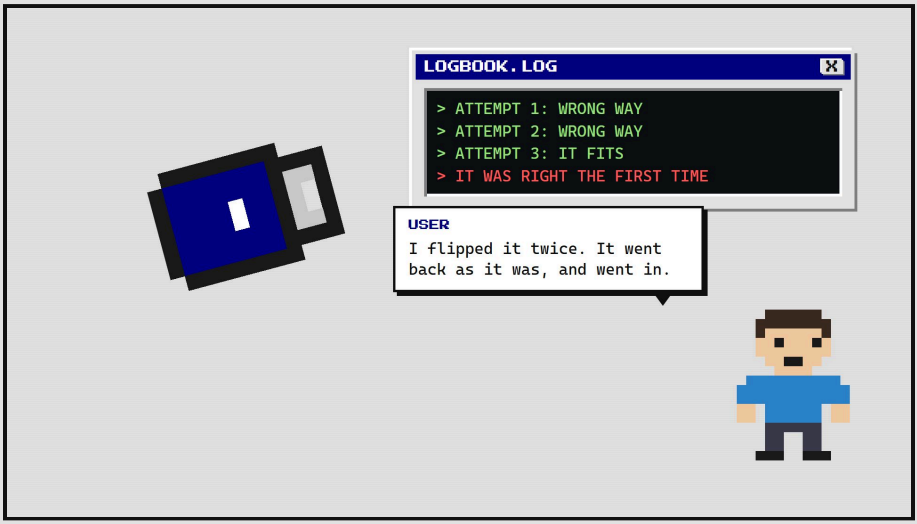
Every home has the Drawer. An ecosystem of tangled cables that breed at night: chargers for phones nobody ever owned, modem extension leads from the last century, a mysterious cable with a round plug that fits nothing.

Fifty cables. Zero with the right connector. Statistically impossible, and yet it happens every time. The cable you need does exist, of course: it's in your neighbour's drawer, and your neighbour is looking for yours.

I proposed a cable exchange to the residents' association, with swaps on Saturday mornings. It failed because nobody wanted to give up their useless cable. ‘You never know,’ they said. The official motto of every drawer.

Real advice: label your cables with a strip of tape and ruthlessly throw away any you haven't recognised for two years.

LAW OF THE QUANTUM USB STICK



A USB plug always goes in on the third attempt: the first time it's upside down, the second time too, and the third time it was the right way round all along.

— Anonymous, *computing folklore*

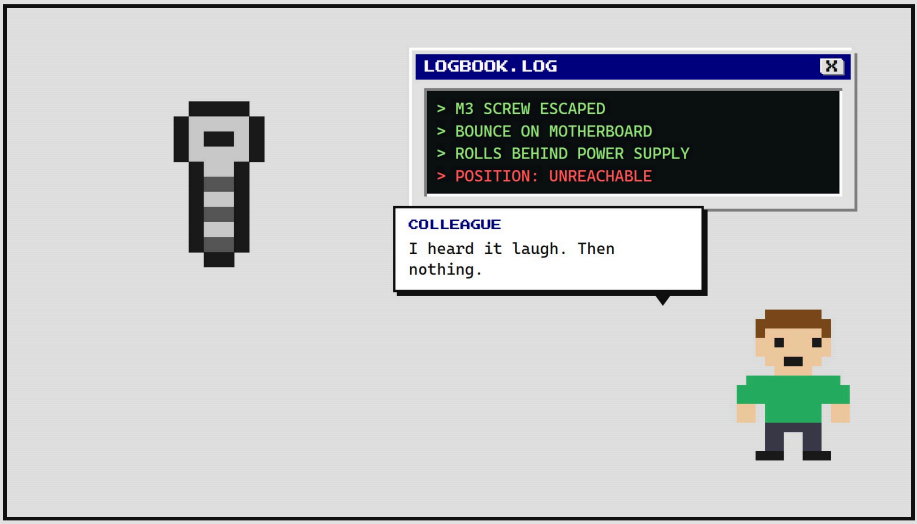
The classic USB connector has two sides. Mathematically, on the first attempt you have a fifty per cent chance of getting it right. In practice, the probability is zero. The connector exists in a state of superposition: it's wrong both ways round until you've turned it twice.

I've calculated that over my career I've spent about three consecutive days of my life turning USB sticks over. I count them as untaken holiday.

Then came the reversible connector, and humanity celebrated. For an afternoon. Then we noticed that reversible cables don't all charge at the same speed, don't all carry data and don't all switch on the monitor. The chaos simply moved inside the cable.

Tip: the USB logo on the connector usually faces upwards. Usually.

LAW OF THE RUNAWAY SCREW



The screw that slips through your fingers while you're opening the computer always falls into the most unreachable spot in the case.

— Anonymous, *computing folklore*

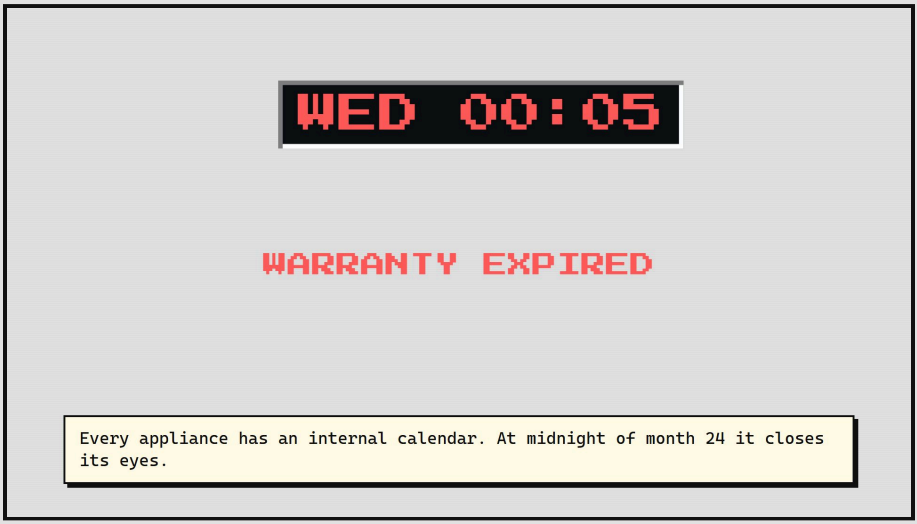
Inside a computer there are thousands of convenient places where a screw could land. It always chooses the only one under the motherboard, behind the power supply, in a gap designed by someone who hated human fingers.

So the technician performs the ancient dance of the case: tilting it, shaking it, turning it upside down, listening. The screw rolls with a metallic sound, like laughter. Then it falls silent. Nobody will ever see it again, but the computer now makes a new noise when you move it.

In my workshop we keep a jar of orphaned screws recovered over the years. Not one has ever been any use. We call it ‘the pension fund’.

A technician's advice: use a magnetic screwdriver and a little bowl for the screws. And never work over carpet.

LAW OF THE EXPIRED WARRANTY



A device breaks down the day after the warranty expires, except when it breaks down the day before and you've lost the receipt.

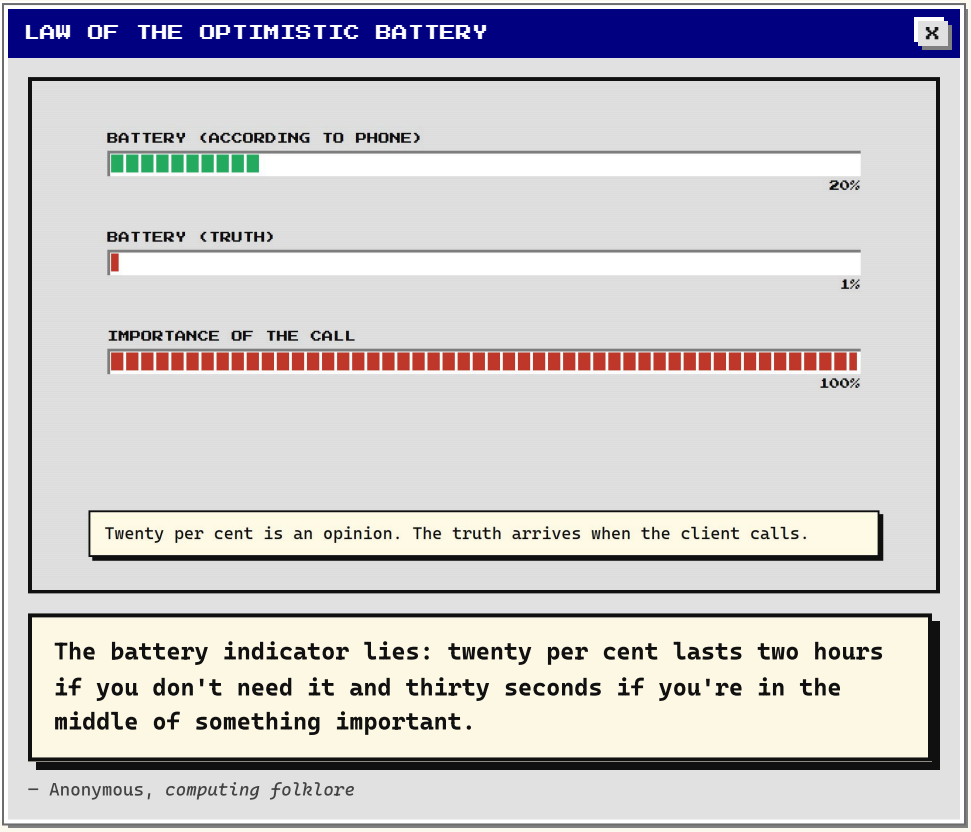
— Anonymous, *computing folklore*

Inside every modern appliance there's a calendar. It isn't in the manual, but it's there. It counts the days with a monk's patience and, at midnight on the twenty-fourth month, closes its eyes in peace.

One of my clients had kept every receipt since 1998 in a shoebox, filed by colour. The day her printer broke down under warranty, the box was in the cellar. The cellar was flooded. The receipt had become a little white cloud.

The warranty, in short, exists: it's the unicorn of customer service. Everyone talks about it, a few swear they've seen it.

A technician's advice: photograph the receipt as soon as you buy something and save it to the cloud. Thermal paper fades, photos don't.



The battery indicator lies: twenty per cent lasts two hours if you don't need it and thirty seconds if you're in the middle of something important.

— Anonymous, *computing folklore*

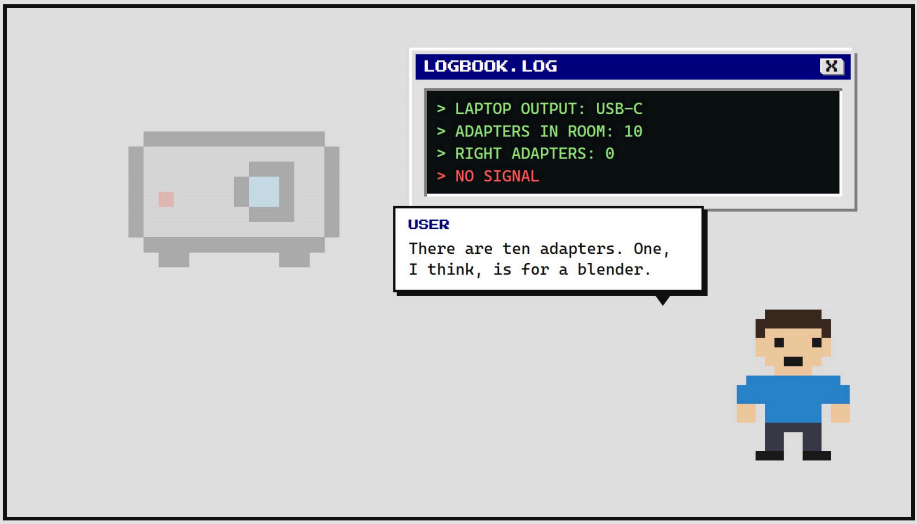
The battery percentage isn't data. It's an opinion. A feeling. A phone on twenty per cent tells you ‘don't worry, we'll make it’, in the tone of someone driving down the motorway with the fuel light on.

Then you open the map to find the client, and the percentage drops like a lift with no cables: eighteen, eleven, four, black. The charger is at home, charging too, out of solidarity.

My uncle maintained that batteries recharge by themselves if you let them rest in a drawer. He's right for about a minute. It's the most optimistic minute of his day.

Real advice: lithium batteries wear out, and as they age the indicator becomes inaccurate. If the device switches off at high percentages, it's time to replace the battery.

LAW OF THE HOSTILE PROJECTOR



The meeting room projector has a different input from your laptop, and the right adapter is the only one nobody brought.

— Anonymous, *computing folklore*

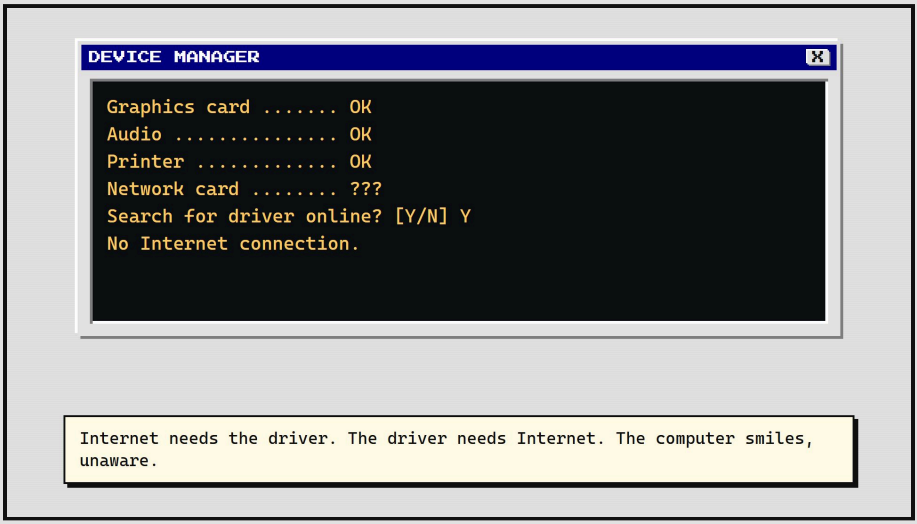
Every meeting room has a projector hanging from the ceiling like an electronic bat. It has a single input, of a type your laptop abandoned three generations ago. On the table, a box of adapters: ten of them, all for other computers, some for objects that aren't computers at all.

Meanwhile the audience waits. Someone suggests ‘emailing the presentation round so everyone can look at it on their phone’. This is the moment civilisation restarts from paper.

When the image finally appears, it's purple, cut off on the right, and the laptop screen goes black. You've won, but you don't know how.

A technician's advice: always bring your own adapter and a PDF copy of the presentation on a USB stick. And arrive fifteen minutes early.

LAW OF THE MISSING DRIVER



After reinstalling the system, the only driver missing is the one for the network card, which you'd need to download all the other drivers.

— Anonymous, *computing folklore*

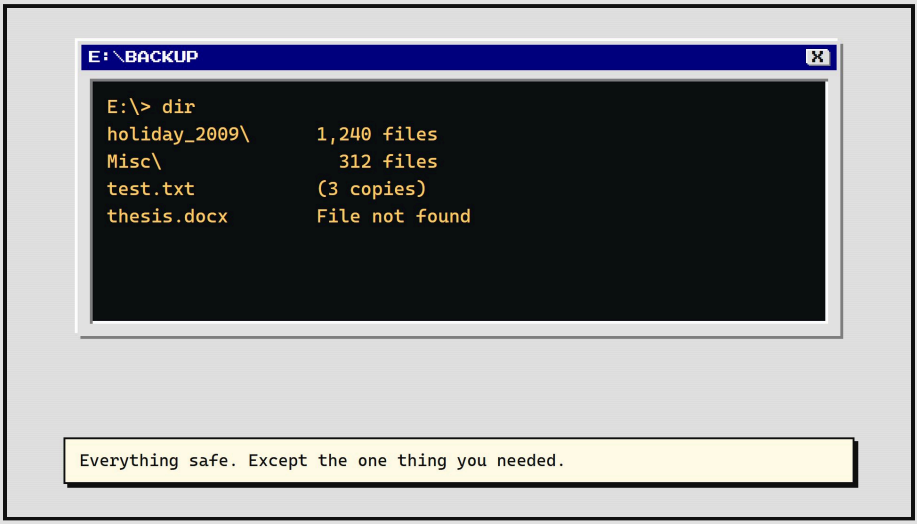
It's the perfect paradox, worthy of a science fiction story. To get the Internet you need the driver. To get the driver you need the Internet. The freshly formatted computer looks at you innocently, like a newborn that can't yet talk to the router.

So the technician sets off on the Great Expedition: another computer, a USB stick, a manufacturer's website that has changed its layout six times. The driver is called something like ‘package_final_v2_new_DEFINITIVE.zip’ and weighs more than the operating system.

Meanwhile, in the next room, the client asks whether, ‘while you're at it’, you could also recover the wedding photos.

Real advice: before formatting, download your model's network drivers onto a USB stick. It's the toolkit for the day after.

LAW OF THE BACKUP YOU DIDN'T MAKE



Of all possible backups, the one you need is always the only one you didn't make.

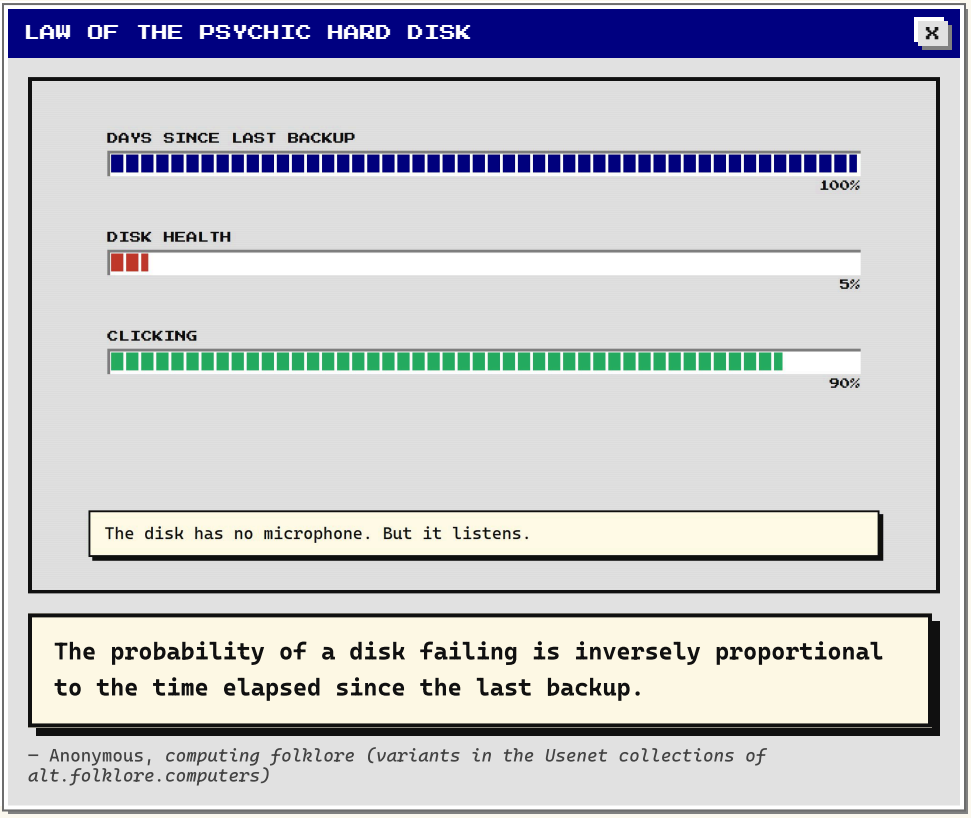
— Anonymous, *computing folklore*

You've got a backup of your 2009 holiday photos. You've got a backup of the 'Misc' folder. You've got three identical backups of a file called 'test.txt'. Then the disk dies, and the only file you really need is your dissertation, which was on the desktop 'just for today'.

A backup is a kind of superstition in reverse: if you make one, nothing happens and you think you've wasted your time. If you don't, the universe arranges a practical demonstration.

I have a client who backed up to a USB stick, which he kept plugged into the computer, which stood under the window, under which a thunderstorm passed. Now he has two wet things instead of one.

Real advice: the 3-2-1 rule. Three copies, on two different media, one of them off-site or in the cloud.



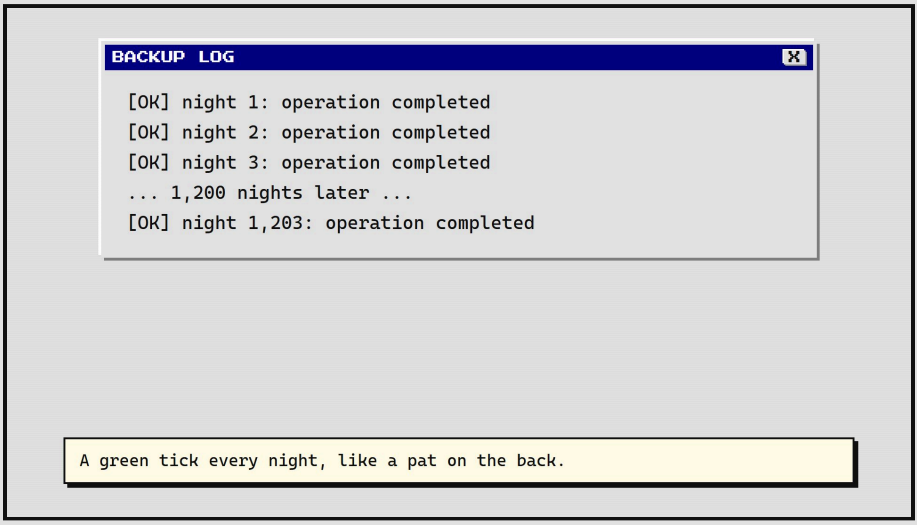
The hard disk listens. It has no microphone, but it listens. The day after a full backup it spins happily, silently, eternally. After six months without copies it starts making a little ticking noise, like a clock in a suspense film.

Old technicians used to say the ticking was ‘the head waving hello’. In fact it was the head waving goodbye for ever.

There's only one way to fool the disk: back up without it knowing. Discreetly, at night, automatically. The disk only discovers the copy afterwards, and by then it no longer feels like breaking. It isn't science. But the results are excellent.

A sysadmin's advice: schedule an automatic backup. A backup that depends on your memory isn't a backup, it's a hope.

LAW OF THE UNTESTED RESTORE



A backup always works, until the day you try to restore it.

— Anonymous, *computing folklore*

For years the backup program showed a green tick. Every night. ‘Operation completed.’ A tick as reassuring as a pat on the back.

Then comes the day of need. You open the archive and discover it contains a single empty folder, copied with religious care for twelve hundred consecutive nights. The program never lied: the operation really was completed. Of what, nobody had asked.

An unverified backup is like a parachute that's never been opened in the shop. It's probably fine. Probably.

Real advice from a technician: every now and then, try restoring a random file from the backup. If you get it back, sleep soundly. If you don't, you've just avoided the worst disaster of all.

LAW OF THE UNSAVED WORK



A program always crashes after three hours of work and thirty seconds before you decide to save.

— Anonymous, *computing folklore*

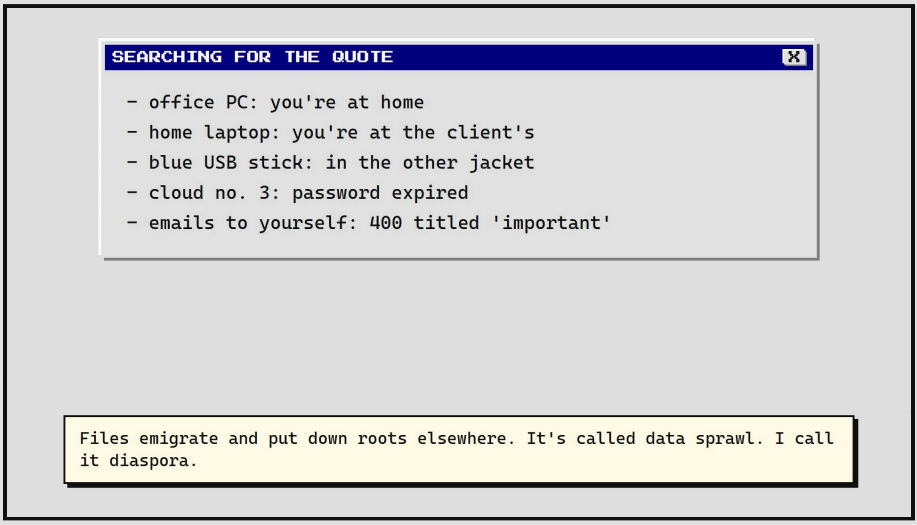
There's a moment, while you're working, when you think: ‘I ought to save.’ It's a noble thought. But you put it off for a minute, just long enough to finish the sentence. The program intercepts that thought, finds it presumptuous, and closes.

It doesn't close badly. It closes elegantly. A little window says ‘The program has stopped working’, like a waiter announcing that the kitchen is closed.

My aunt saved every document after every word. It took her an afternoon to write a birthday card, but she never lost anything. We called her ‘the Safe’.

Real advice: learn the Ctrl+S combination and use it like a nervous tic. And turn on autosave, if the program has it.

LAW OF THE FILE ON THE OTHER COMPUTER



The file you urgently need is always on the other computer, the other USB stick or in the other house.

— Anonymous, *computing folklore*

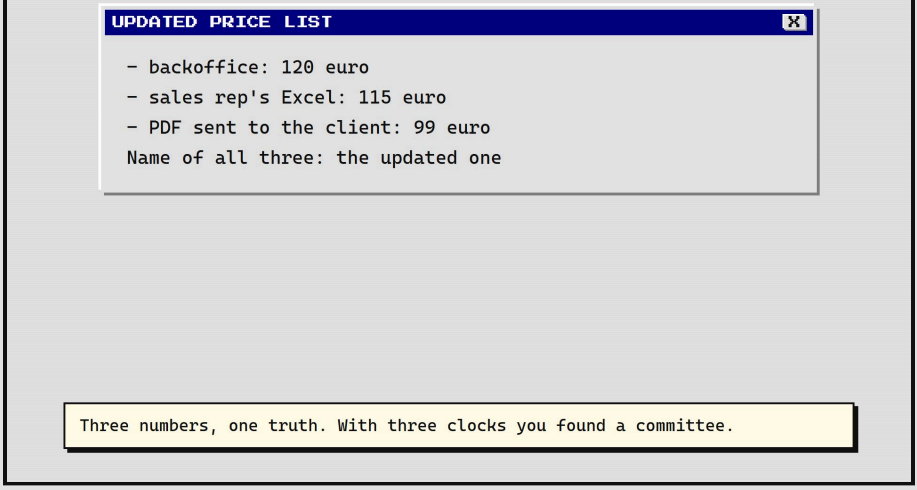
You have three computers, two USB sticks, an external drive and four cloud services. The quote file is in just one of those places. And it's always the one you can't get at: the office computer when you're at home; the home laptop when you're at the client's.

It's called data sprawl, but I call it the diaspora. Files emigrate, put down roots elsewhere, change their names to ‘quote_final_2_copy’.

Once I spent an hour looking for a document. It was attached to an email I'd sent myself with the subject line ‘important’. There were four hundred with the same subject.

Real advice: choose a single main place to work, synchronised, and give files clear names with a date. Your future self will thank you.

SEGAL'S LAW



A man with a watch knows what time it is. A man with two watches is never sure.

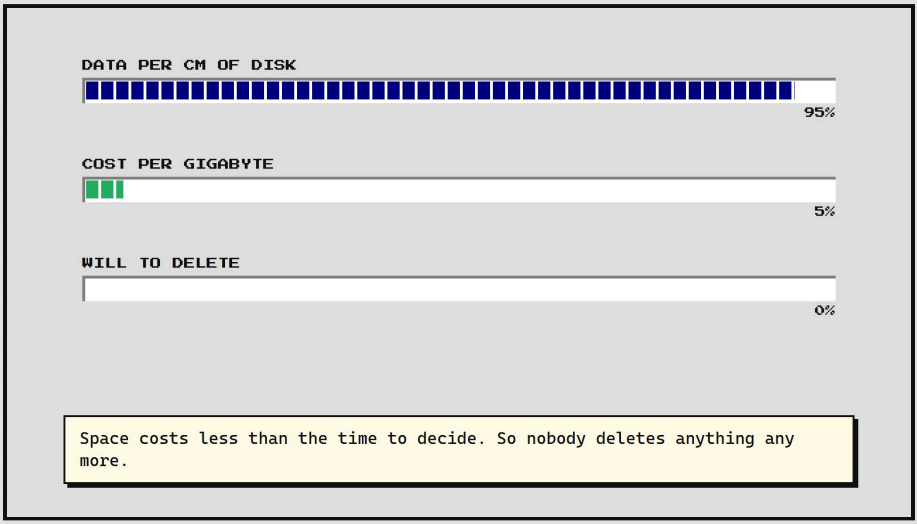
— Anonymous, *Saying that appeared in the San Diego Union in 1930; later misattributed to Lee Segall and made famous as 'Segal's Law' in Arthur Bloch's books (uncertain attribution)*

With a single watch you have no doubts: you're wrong with complete serenity. With two watches showing different times, the crisis begins. Which one is right? With three, you set up a committee.

In computing it's the drama of multiple sources. The price list lives in the management system, in the salesman's Excel sheet and in the PDF sent to the client. They're three different numbers, and all of them are called 'the up-to-date one'. That's why people who design serious systems talk about a single source of truth: one place everyone reads from, and the others copy.

It applies to servers too: if the clocks of the computers on a network aren't synchronised, the logs tell impossible stories, with answers arriving before the questions. I've solved it at the office. I removed every clock but one. It stopped in 2016, but nobody argues any more.

KRYDER'S LAW



The density of data that can be recorded on a magnetic disk grows faster than the power of chips, and the cost per gigabyte collapses as a result.

— Mark Kryder, *'Kryder's Law' article by Chip Walter, Scientific American, August 2005*

The first commercial hard disk weighed as much as a fridge and held a few megabytes. Today a card the size of a fingernail holds all your holiday photos, including the forty identical ones of the sunset. That's Kryder's Law: storage gets denser and cheaper at a dizzying pace.

The psychological consequence is devastating. Nobody deletes anything any more. Why choose, when space costs less than the time it takes to decide? I have clients with twelve copies of the same PDF, called final, final2, final_really and final_really_this_time.

In recent years growth has slowed, but humanity has already found the solution: hoard anyway. My uncle keeps his broken disks in a box. He says the data is still in there, just a bit shy.

THE SYSADMIN'S ADVICE

Two things almost nobody does and everybody should. The first: **ask the disk how it's doing**. Almost all disks, mechanical and SSD, keep a log of their own health called SMART. On Windows a free program such as CrystalDiskInfo reads it in a moment and sums it up with a colour: blue is fine, yellow is a warning, red is trouble. Yellow is not an opinion, it is advance notice: copy your data that same day. SSDs, unlike old disks, often fail without any farewell clicking.

The second: **don't trust hibernating disks**. An external disk shut in a drawer for years may not start again, and forgotten USB sticks lose bits. Power up your archives now and then, check that the files open and every few years copy everything onto a new medium. Data doesn't age. Its containers do.

CHAPTER 3

CH 03 - HELPDESK

Helpdesk: users, passwords and other mysteries

This is the chapter of my everyday work, and so the one I reread with the most emotion. Here are the ringing phone, the user who didn't touch anything, the error message closed at the speed of light, the computer that heals as soon as the technician crosses the threshold.

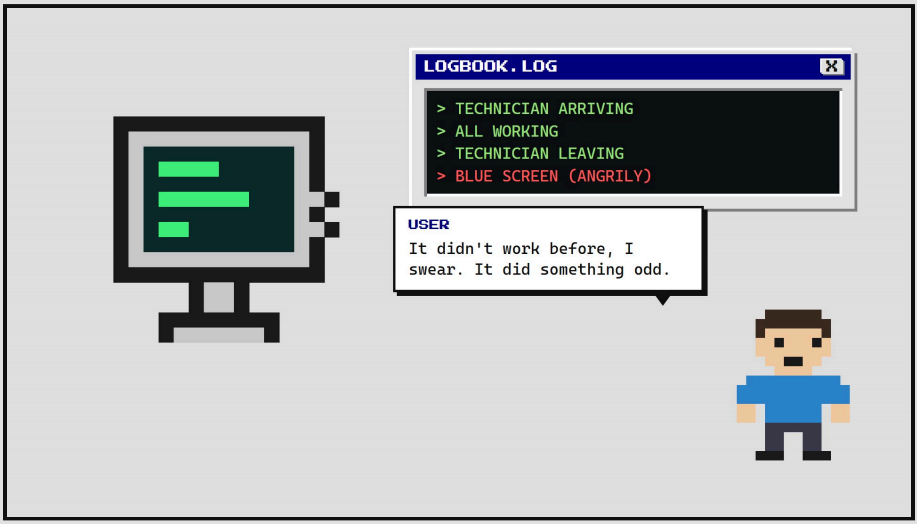
Let me make one thing clear straight away: here we don't laugh at users. We laugh with them. Every technician is a user of something else, and in front of my boiler at home I am exactly that gentleman who swears he didn't touch anything. Hanlon's razor, in the first chapter, says it better than me: behind a disaster there is usually no malice. There is a click.

Then there are passwords, which deserve a section of their own: too simple, too complicated, forgotten the next day. And the chapter closes with a few serious design laws, the ones that explain why a small, distant button gets missed, why a menu with forty items paralyses and why half a second of waiting feels like an eternity. Reading them helps you see that many ‘user errors’ are really errors by whoever designed the screen.

CONTENTS.DIR

01 Law of the healing technician
02 Law of ‘I didn't touch anything’
03 Law of the unplugged plug
04 Law of the unread error message
05 Law of the Friday night failure
06 Law of the relative who knows about computers
07 Law of the forgotten password
08 Law of the never-good-enough password
09 Kerckhoffs's principle
10 Gilb's Laws of Unreliability
11 Tesler's Law
12 Jakob's Law
13 Fitts's Law
14 Hick's Law
15 The Doherty threshold

LAW OF THE HEALING TECHNICIAN



The problem disappears at the exact moment the technician arrives, and reappears as soon as they leave.

— Anonymous, *computing folklore*

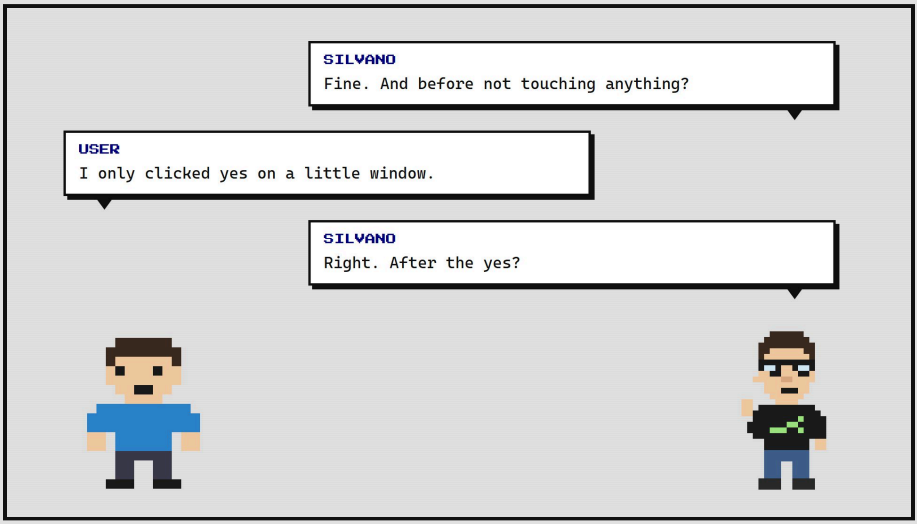
It's my only superpower. I walk into the room, and the computer that has been showing blue screens for three days stands to attention like a soldier in barracks. ‘It wasn't working before, I swear,’ says the user. I believe them. I've seen this scene a thousand times.

The problem recognises the technician the way a dog recognises the vet: it behaves, wags its tail, denies everything. Then, as my car leaves the car park, the computer starts again, a little angrier about the time it's lost.

I've thought about leaving a life-size photo of myself at the office. One client did it with a cardboard cut-out. He says it works for about a week.

A technician's advice: when the error appears, take a photo of the screen with your phone. One photo is worth a thousand ‘it did something strange earlier’.

LAW OF ‘I DIDN'T TOUCH ANYTHING’



The user who says ‘I didn't touch anything’ touched something; the only unknown is what.

— Anonymous, *computing folklore*

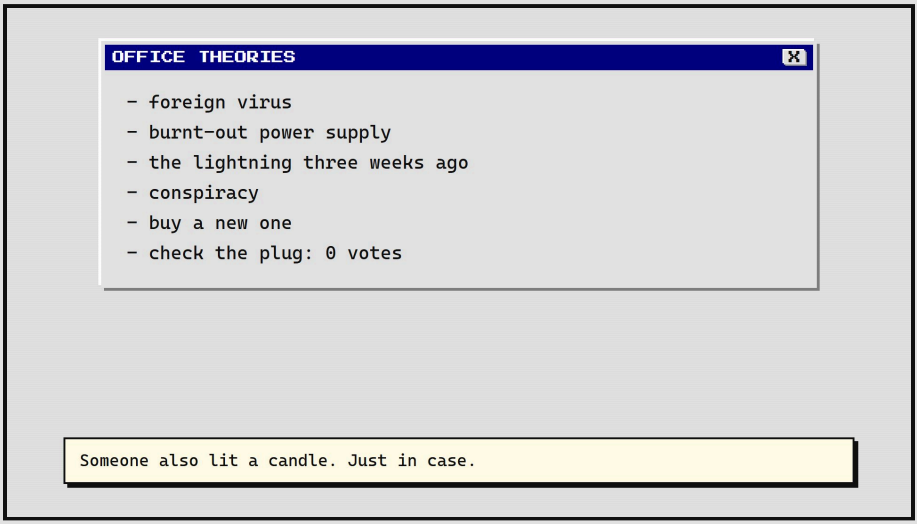
‘I didn't touch anything’ is the oldest sentence in technical support. It was found carved on the first valve computer, next to a fingerprint.

The user isn't lying. In their memory they didn't touch anything. They just clicked ‘yes’ on a little window, moved a folder ‘that was in the way’, installed a free program to see their photos better and changed a small setting that ‘looked ugly’. But touch anything? No.

The experienced technician doesn't argue. He nods with the serenity of a confessor and asks: ‘All right. And before you didn't touch anything, what were you doing?’ That's when the truth comes out.

Real advice: describing your last few steps honestly saves hours. The technician isn't judging; they just need to know where to start.

LAW OF THE UNPLUGGED PLUG



Before anyone checks whether it's plugged in, a virus, a motherboard failure and a conspiracy will already have been suspected.

— Anonymous, *computing folklore*

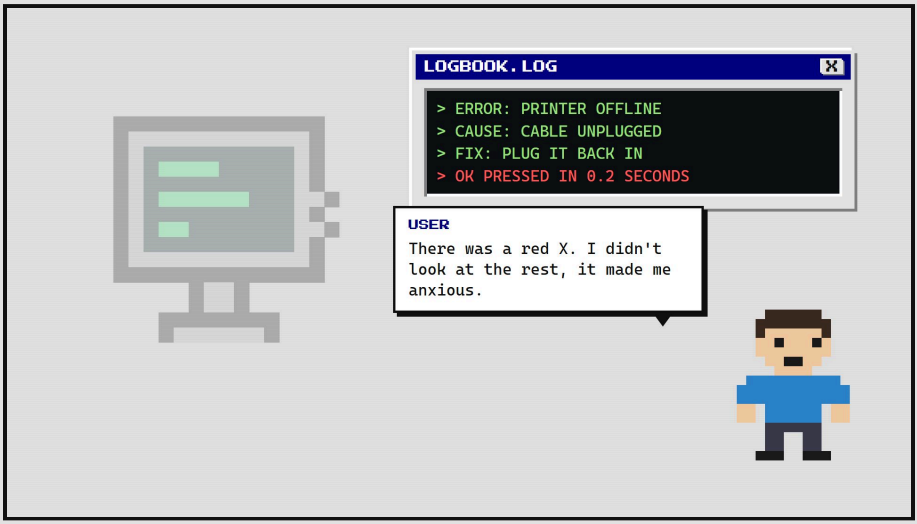
The computer won't switch on. Immediately the office splits into schools of thought. Some talk of foreign viruses, some of a burnt-out power supply, some of the lightning strike three weeks ago that ‘must have left something behind’. Buying a new computer is considered. A candle is lit.

Then the technician arrives, bends down and plugs back in the plug the cleaning company had pulled out to use the hoover. The silence that follows is sacred.

The unplugged plug is the humblest cause in computing. It has no error code, no log, no glamour. But it always wins.

A technician's advice: check the obvious things first. Plug, power strip switched on, monitor cables pushed in properly. It isn't trivial: it's method.

LAW OF THE UNREAD ERROR MESSAGE



The user closes the error message before reading it, then calls the technician to find out what it said.

— Anonymous, *computing folklore*

The error message is a misunderstood little poet. It appears with all the necessary information: the problem, the cause, even the solution. The user looks at it with the horror of someone finding a spider in the bath, and presses ‘OK’ at a speed that defies physics.

Then they ring: ‘A message came up.’ ‘What did it say?’ ‘I don't know, it was in English. Or Italian. There was a red X.’ That's all that remains, like an account of a UFO sighting.

So the technician has to recreate the error, waiting for the poet to come back and recite. The poet, offended, does not come back.

Real advice: when an error appears, don't close it straight away. Read it calmly, take a photo or a screenshot. Half the time the answer is already there.

LAW OF THE FRIDAY NIGHT FAILURE



Serious faults only occur when the technician isn't around: on Friday evening, on the August bank holiday or on the first day of the holidays.

— Anonymous, *computing folklore*

The server has worked without a murmur for eleven months and three weeks. Then the technician books a hotel by the sea. The server knows. Don't ask me how: it knows.

On Friday at one minute past six, as the keys turn in the office lock, a little light turns orange. At two minutes past, red. At three minutes past, the technician's phone rings, and he's already on the motorway with the family and an inflatable lilo on the seat.

I once knew a sysadmin who hadn't taken a holiday for six years for fear of this law. The server never broke down. Then he went to the dentist on a Tuesday morning.

Real advice: clear documentation, access details recorded somewhere safe and a backup colleague. Being on call isn't heroism, it's organisation.

LAW OF THE RELATIVE WHO KNOWS ABOUT COMPUTERS



Whoever in the family ‘knows about computers’ will spend every holiday lunch setting up an uncle's printer.

— Anonymous, *computing folklore*

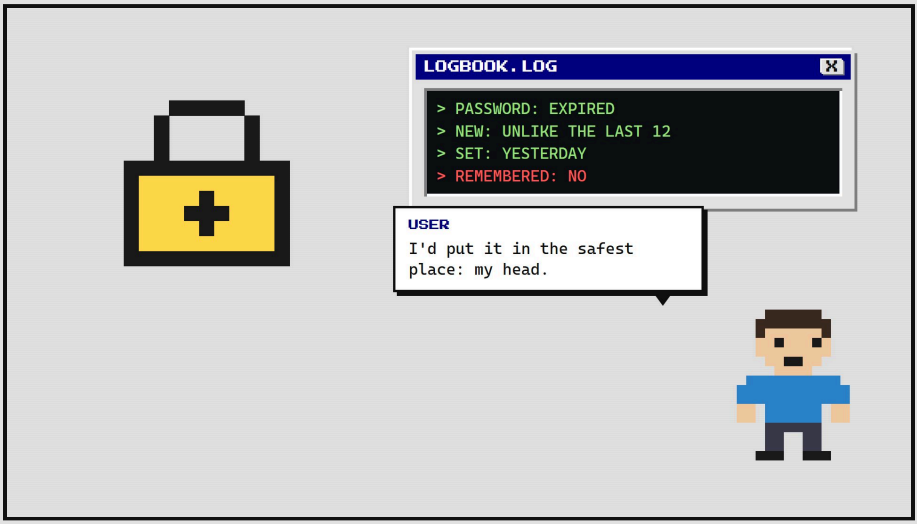
It only takes once. Just one time when you sort out Grandma's phone, and you're appointed Family IT Manager for life. No salary, no holidays, permanently on call on bank holidays.

You arrive for Christmas lunch and haven't even taken your coat off. ‘While you're here, could you have a look at the computer?’ The computer is in the loft, it's twelve years old and has a film of dust that acts as thermal insulation.

By the end of the day you've eaten a cold roast potato, updated four phones and explained six times that the Wi-Fi is not ‘the Internet’. They pay you with a jar of jam. It's excellent, let's be honest.

Real advice: write down the Wi-Fi password and the passwords for your relatives' main accounts on a sheet of paper, and leave it with them somewhere safe. You'll halve the phone calls.

LAW OF THE FORGOTTEN PASSWORD



A password is always forgotten the day after you changed it, because the system made you.

— Anonymous, *computing folklore*

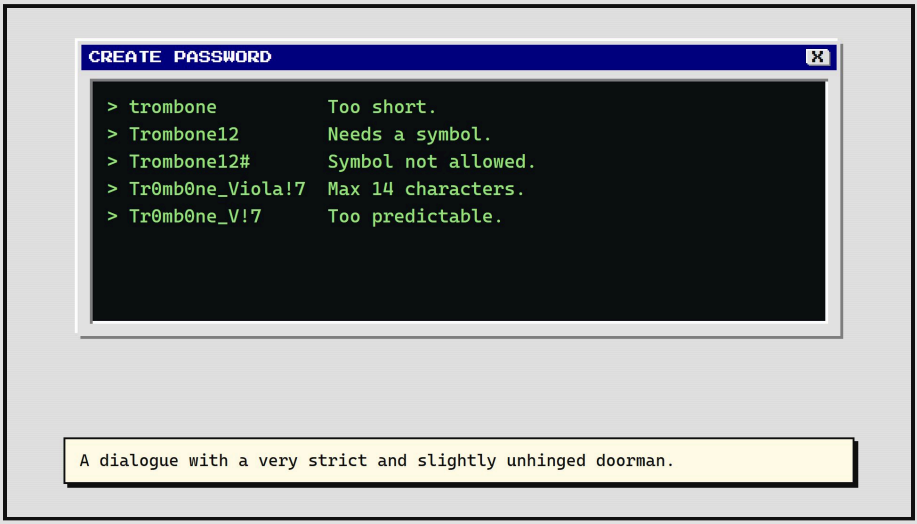
For three years you used the same password. You knew it like your child's name. Then the system decides it has expired and wants a new one, different from the last twelve. You come up with it carefully. The next day it has vanished from your memory like a dream on waking.

So begins the hunt for the reset. ‘Enter the answer to your security question.’ The name of your first pet. Did you write ‘Fluffy’, ‘fluffy’ or ‘FLUFFY’? The system won't say. It locks you out for thirty minutes to think about it.

The yellow sticky note under the keyboard, meanwhile, is laughing. It was there all along.

Real advice: use a password manager. You remember just one strong password, and it looks after the rest. The sticky note does not count as a manager.

LAW OF THE NEVER-GOOD-ENOUGH PASSWORD



The website rejects every password until it becomes impossible to remember, and then rejects that one too because it contains a character that isn't allowed.

— Anonymous, *computing folklore*

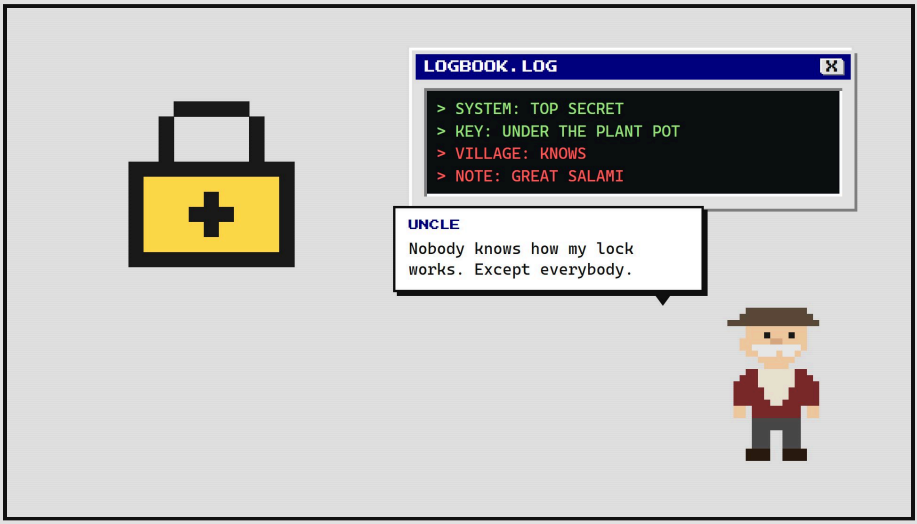
‘Your password must contain at least eight characters.’ Done. ‘One capital letter.’ Done. ‘One number.’ Done. ‘One symbol.’ Done. ‘It cannot contain the symbol you used.’ Ah.

It's a conversation with a very strict and slightly unhinged doorman. Every attempt unlocks a new rule, like a video game whose final level was written by a committee.

In the end you get a sequence like ‘Tr0mb0ne_Purple!7’, proud as a flag. You write it on a sheet of paper. The paper goes in a drawer. The drawer has no lock. Cybersecurity has come full circle and is back where it started.

Real advice: a long phrase of random words is better than a short word full of symbols. It's more secure and easier to remember.

KERCKHOFFS'S PRINCIPLE



A cryptographic system must remain secure even if the enemy knows everything about how it works: the only secret must be the key.

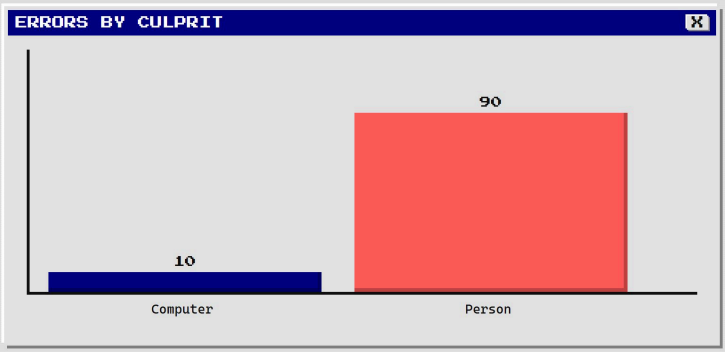
— Auguste Kerckhoffs, *La cryptographie militaire*, *1883*

The opposite is called 'security through obscurity': the program is secure because nobody knows how it works. It lasts precisely until the first curious person with a free afternoon.

Kerckhoffs reasoned like a soldier: devices get captured, manuals copied, spies bribed. If all the security lies in the secrecy of the mechanism, one theft and you lose everything. If it lies in the key, you change the key and start again. That's why serious algorithms are public and studied by everyone.

My uncle protects his cellar with a top-secret system: the key under the geranium pot. The system is obscure, but the whole village knows it. On Sunday someone left him a note: 'Excellent salami, good vintage.' Security through geraniums: not recommended.

GILB'S LAWS OF UNRELIABILITY



The computer always errs the same way. People err with creativity.

Computers are unreliable, but humans are even more unreliable; and any system that depends on human reliability is unreliable.

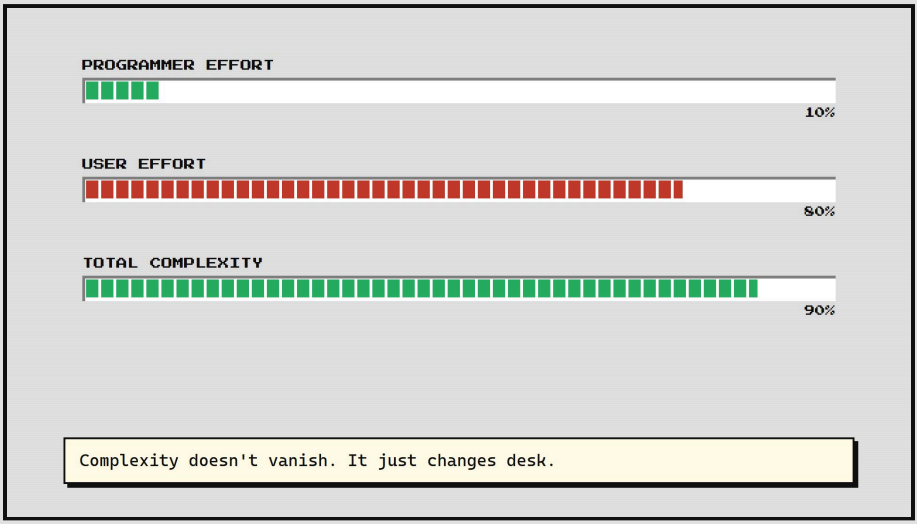
— Tom Gilb, *Laws circulated in 1970s collections (e.g. Arthur Bloch); numbering and wording vary between sources (attribution partly uncertain)*

Computers make mistakes, yes. But they always make them the same way, and sooner or later you work it out. Humans make mistakes creatively: today the password on a sticky note, tomorrow the sticky note on the screen, the day after the screen photographed and shared in the family group chat.

The lesson is an engineer's: don't design systems that only work if everyone is paying attention. People are tired, distracted, in a hurry. Automatic checks, confirmations and backups exist for them.

One company had a foolproof backup procedure: every evening an employee swapped the disk. It worked for years. Then the employee retired. The disk is still there, inserted since 2015, as faithful as a dog waiting at the station.

TESLER'S LAW



Every application has an amount of complexity that cannot be removed: you can only decide whether the programmer or the user has to deal with it.

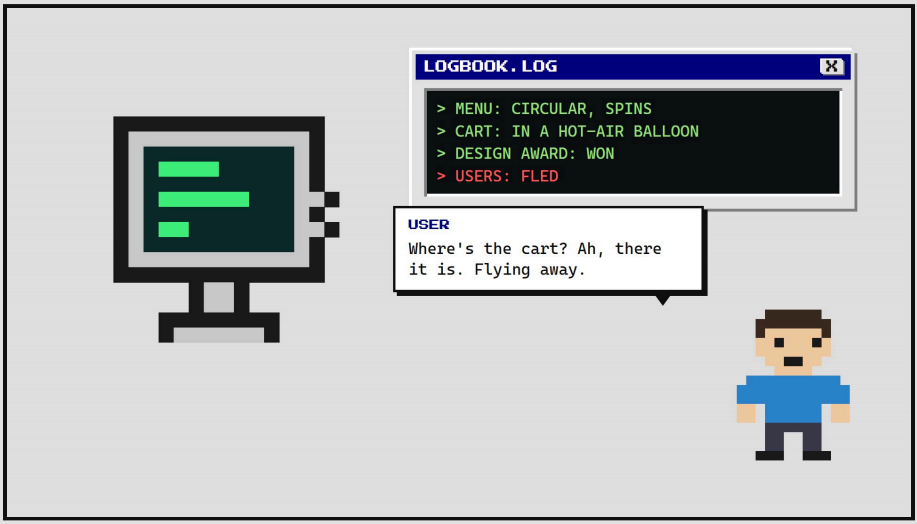
— Larry Tesler, *Law of conservation of complexity, formulated in the 1980s during his work on interfaces, including at Apple*

Complexity is like energy: it is neither created nor destroyed, it just moves. A form with a single 'simple' field means that somebody behind it wrote a thousand lines to work out what the user meant.

The law is an invitation to courtesy: better for one developer to suffer once than for ten thousand users to suffer every day. Unfortunately the opposite temptation is strong, and so the form with forty mandatory fields is born.

At the office we simplified the printer: it now has a single button. All the complexity has been transferred to me, and I get twenty-two phone calls a day explaining what that button does. Conservation works. I work rather less well.

JAKOB'S LAW



Users spend most of their time on other websites, so they expect yours to work the same way as the ones they already know.

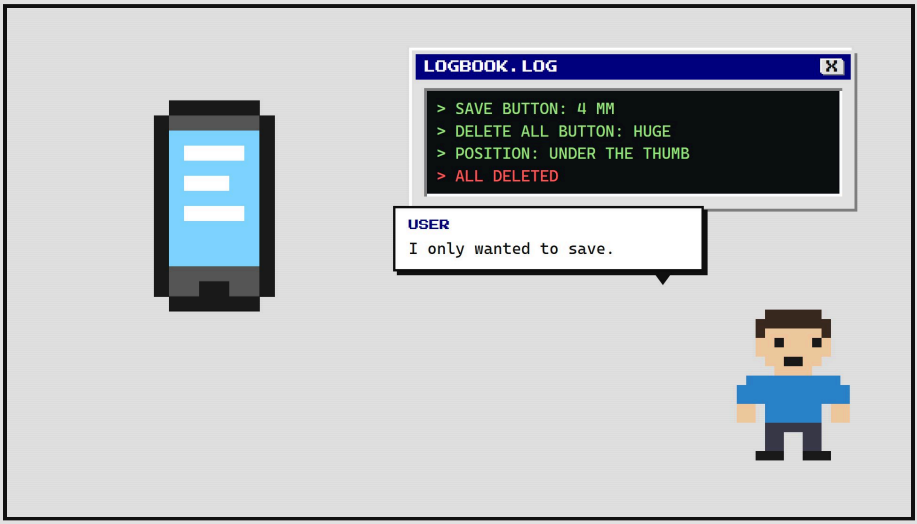
— Jakob Nielsen, *Alertbox, Nielsen Norman Group, 2000*

The basket is top right. The logo top left takes you back to the home page. The magnifying glass searches. Nobody passed a law about it, but millions of people have learnt it by using it everywhere. Jakob's Law says: don't fight these habits, ride them.

Every so often a visionary designer turns up wanting to astonish. A circular menu that rotates as you scroll. A buy button hidden in an animated hot-air balloon. The site wins an award. Users flee to a boring site where the basket is where it belongs.

Being original is fine for colours, photos and copy. In navigation, originality is a cost the user pays. Once I put the menu bottom left for artistic reasons. The client found it after a week. He'd come to ask me why his website only had one page.

FITTS'S LAW



The time needed to reach a target with a movement depends on its distance and size: the further away and smaller it is, the longer it takes and the more often you miss.

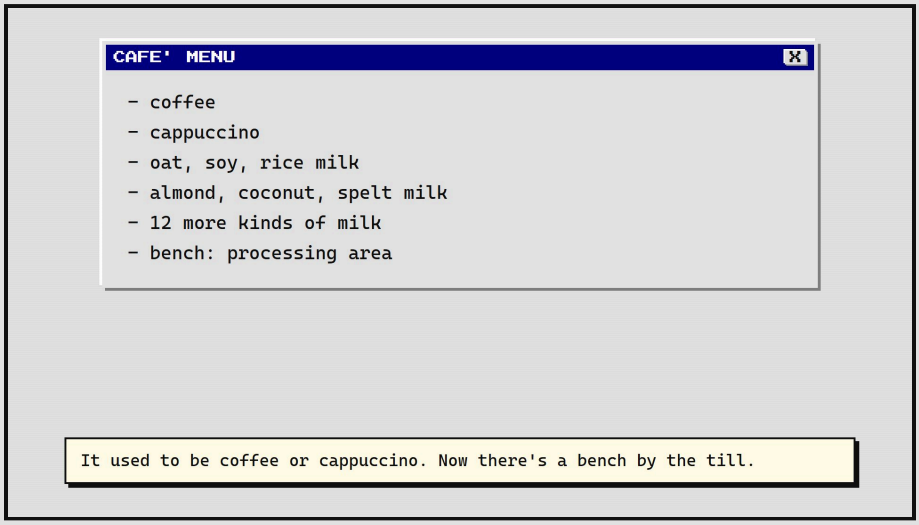
— Paul Fitts, *Journal of Experimental Psychology*, 1954

A psychologist in the 1950s studied hand movements, long before mice existed. His formula, though, seems written for interface designers: a big, nearby button is clicked quickly; a small, distant one is a funfair test of skill.

That's why important buttons should be big, and dangerous ones far away. Instead the opposite often happens: the 'Save' button is the size of a postage stamp, and 'Delete all' is as big as a door, right under your thumb. On a phone it becomes an extreme sport.

The edges of the screen are infinite targets, because the mouse stops there by itself. My uncle proves it: he always opens the Start menu by flinging the cursor to the bottom left with a flick of the wrist. He says it's a fishing technique. He never misses.

HICK'S LAW



The time needed to make a decision increases with the number of available options (logarithmically, not linearly).

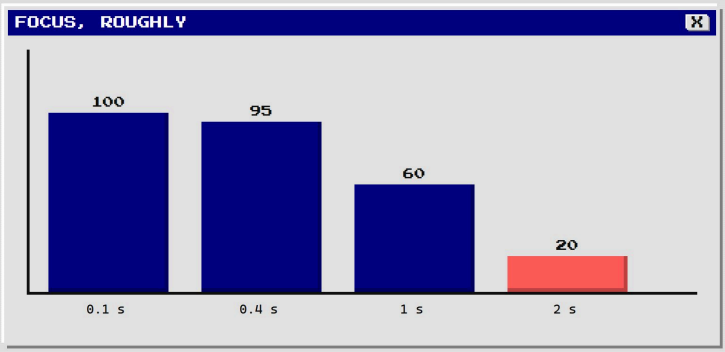
— William Edmund Hick (and Ray Hyman), *Quarterly Journal of Experimental Psychology, 1952; Hyman's study, 1953 (also called the Hick–Hyman law)*

A menu with three items is read and chosen from in an instant. A menu with forty items, each with a submenu, is an afternoon of your life. Hick's Law says that the more options you put in front of someone, the longer they take to decide. It grows less than proportionally, but it grows.

It's why good programs hide the advanced settings and show only what's needed. And it's why the options window of certain software, with thirty-two tabs and two hundred checkboxes, gives the user a fixed stare towards the horizon.

I see it every day at the café. It used to be 'coffee or cappuccino'. Now there's a menu with eighteen kinds of milk. The barista has put a bench in front of the till. He calls it the processing area.

THE DOHERTY THRESHOLD



Above the threshold the mind starts to wander. And when it wanders, it goes online.

Productivity rises sharply when the computer responds to the user in less than about 400 milliseconds: below that threshold, neither one is waiting for the other.

— Walter J. Doherty and Arvind J. Thadhani, *'The Economic Value of Rapid Response Time', IBM, 1982 (the 400 ms figure is cited mainly by later sources)*

In the early 1980s a two-second wait was thought acceptable. An IBM study argued otherwise: if the computer responds in under roughly half a second, the user stays focused and gets much more done. Above that, the mind starts to wander. And when it wanders, it wanders onto the Internet.

That's why modern interfaces show something straight away: an animation, a bar, a grey skeleton of the page. It isn't faster, but it looks alive. Users don't want the result instantly. They want to know someone is listening.

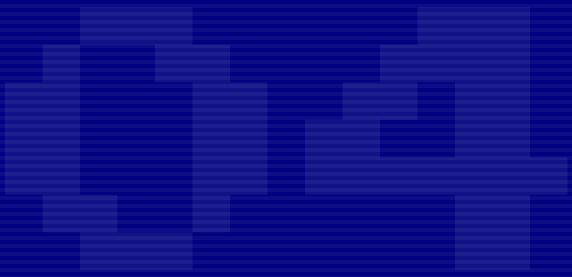
I've applied the Doherty threshold to phone support. I answer within four hundred milliseconds with 'Yes, I'm here.' Then, obviously, I take an hour to fix it. But the client feels looked after. It's all in the emotional progress bar.

THE SYSADMIN'S ADVICE

Most support time goes into understanding what happened. You can give that time to your technician as a present, and they will remember you fondly. When something goes wrong, have these five pieces of information ready before you call:

- **What you were doing**, step by step, including the ‘I didn't touch anything’ in detail.
- **The exact message**, photographed. On Windows 10 and 11 the Windows + Shift + S keys capture part of the screen.
- **Since when** it has been happening, and whether it used to work.
- **What has changed** in the meantime: an update, a new program, a desk move.
- **Whether it happens always** or only sometimes, on one computer or on all of them.

With these answers, many faults are fixed over the phone. The others are fixed faster, because the technician arrives with the right tool already in hand.



CHAPTER 4

CH 04 — NETWORK

Networks and the Internet: the invisible wire

The network is the most used and least understood thing in the modern home. Everyone wants it, nobody knows where it is. In the router? In the wall? In the air? My neighbour maintains that it comes through the phone socket and rests at night. I didn't contradict her: in some neighbourhoods she's right.

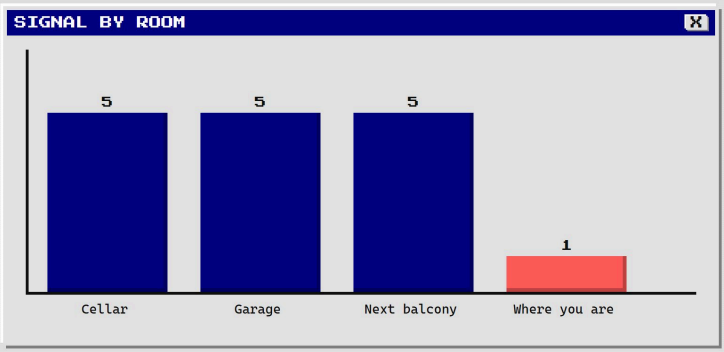
The chapter opens with the laws of the home and office connection: the Wi-Fi that works everywhere except where you are, the download that dies at ninety-nine, the video call that freezes when it's your turn. Then it moves on to the laws of the wider Internet, born among forums, online newspapers and group chats. Here technology matters little and people matter a great deal: those who correct and get it wrong, those who put a question mark in the headline, those who try to hide something and make it famous.

These laws also apply away from the screen. Brandolini's Law, for instance, I have seen at work over dinner, at a village fête and once in a waiting room: refuting nonsense always takes more effort than saying it. This chapter doesn't refute it. It documents it.

CONTENTS.DIR

01 Law of the spiteful Wi-Fi
02 Law of the router reboot
03 Law of the crucial video call
04 Law of the download at 99%
05 Law of the website that works everywhere
06 Metcalfe's Law
07 Nielsen's Law of bandwidth
08 The robustness principle
09 Cunningham's Law
10 Skitt's Law
11 Betteridge's law of headlines
12 Poe's Law
13 Godwin's Law
14 The Streisand effect
15 Brandolini's Law

LAW OF THE SPITEFUL WI-FI



Signal bars. Wi-Fi is a cat: it goes where it pleases.

The Wi-Fi works perfectly in every room of the house except the one you're in.

— Anonymous, *computing folklore*

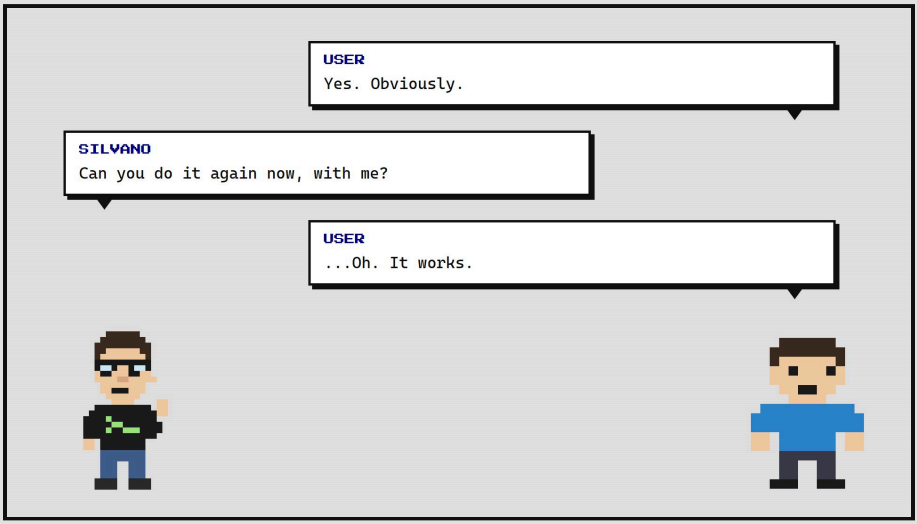
The Wi-Fi signal is a cat. It goes where it pleases. In the cellar it's excellent, in the garage too, on next door's balcony it's high definition. But you're in the lounge, and in the lounge there are two bars flashing like a cry for help.

So the ritual begins. You move half a metre. You raise the phone to the sky like an offering. You perch on the arm of the sofa in a position your physiotherapist would advise against. It works. You move a centimetre. It stops working.

I have a client who put the router inside a solid wood wardrobe ‘so as not to see it’. Now the Wi-Fi can't see it either.

A technician's advice: the router should go in a central, high, unobstructed position. Thick walls, mirrors and aquariums are its natural enemies.

LAW OF THE ROUTER REBOOT



Before calling support, nobody reboots the router; after calling, the problem is solved by rebooting the router.

— Anonymous, *computing folklore*

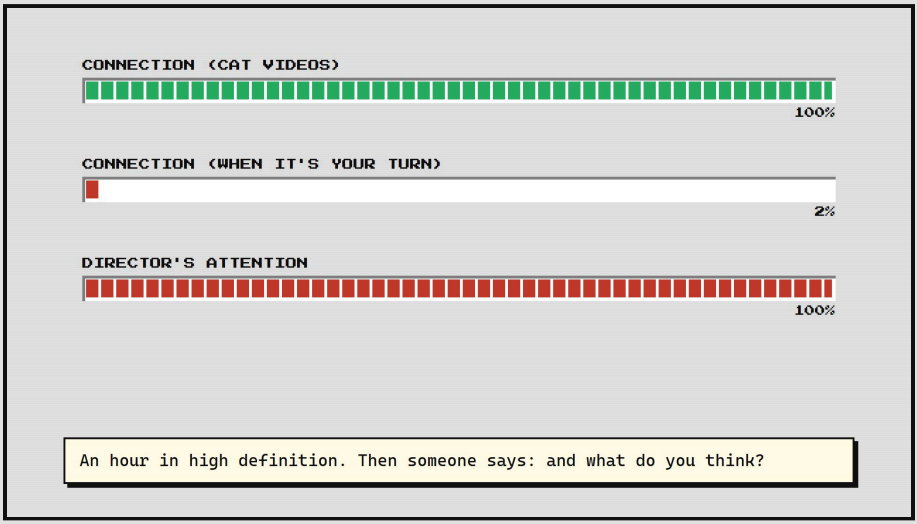
Rebooting the router is the secular sacrament of the digital age. Everyone knows its power, and nobody really believes in it until someone with a headset and a freephone number says it.

‘Have you tried turning it off and on again?’ ‘Of course.’ ‘Could you do it again now, with me?’ Silence. Sound of furniture being moved. Little lights going off. Little lights coming on. ‘Oh. It works now.’

Brothers and sisters of connectivity! The router is not a device. It is a state of mind that needs resetting every so often, like us on a Monday morning.

A technician's advice: switch it off, wait at least thirty seconds, then switch it back on and give the router a couple of minutes to come back up. If the problem keeps returning, look for the cause: a reboot is a sticking plaster, not the cure.

LAW OF THE CRUCIAL VIDEO CALL



The connection holds up through hours of cat videos, but drops exactly when it's your turn to speak in the important video call.

— Anonymous, *computing folklore*

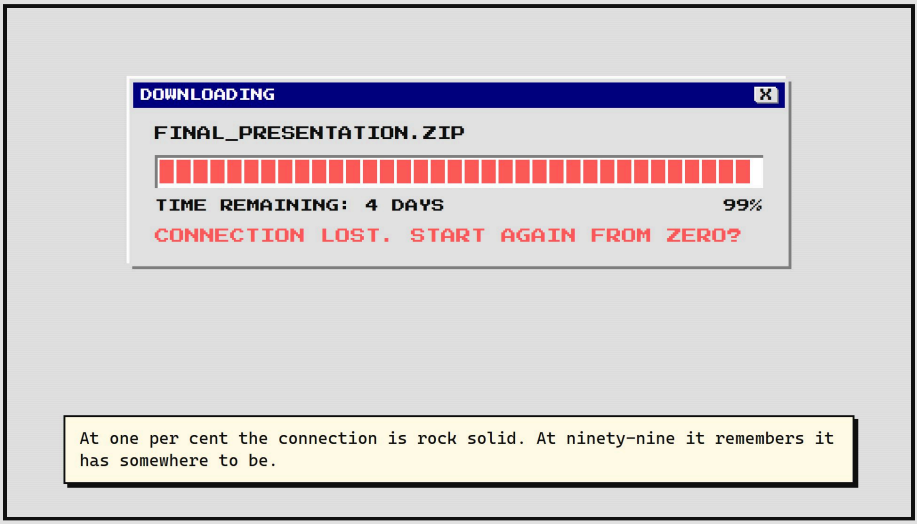
All through the meeting the connection has been flawless. You've seen your colleagues in high definition, heard the director breathing, counted the plants behind his desk. Then somebody says: 'And what do you think?'

At that precise instant your face freezes in an unreadable expression, halfway between astonishment and a sneeze. Your voice becomes that of a robot underwater. The others see you like that for forty seconds, and decide to move on.

By the time you're back online, the meeting has already decided. About you. In your absence.

Real advice: for important calls, if you can, connect the computer to the router with a network cable. Wi-Fi is convenient; the cable is reliable.

LAW OF THE DOWNLOAD AT 99%



At one per cent the connection is rock solid. At ninety-nine it remembers it has somewhere to be.

A download or upload always fails at ninety-nine per cent, never at one.

— Anonymous, *computing folklore*

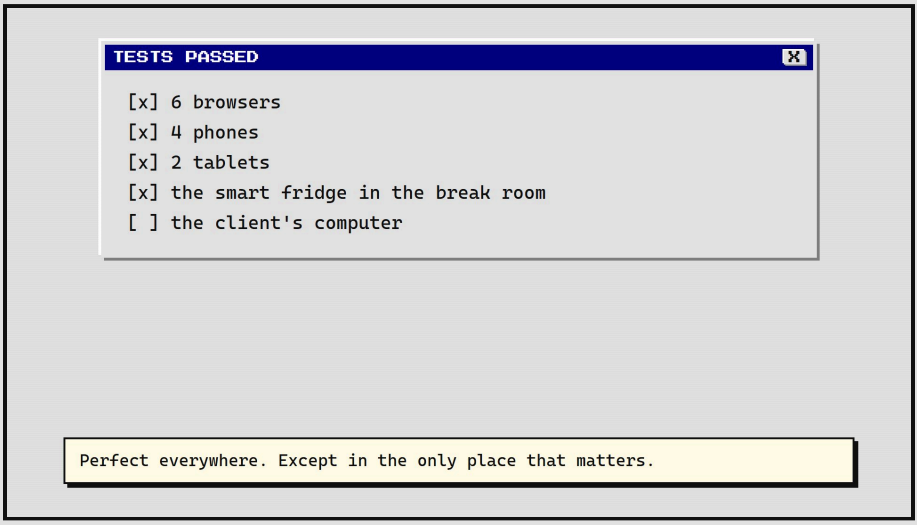
No download has ever failed at one per cent. That would be an honest failure, painless. No: the bar creeps along for an hour, reaches ninety-nine, gives you a glimpse of the promised land, and then the words ‘network error’ appear.

Scientists call it cruelty. Technicians call it ‘the last one per cent’, which famously lasts as long as the other ninety-nine. That's where the computer does the serious calculations, they say. In reality it's just deciding whether it likes you.

I've seen grown men stare at a bar stuck at ninety-nine for twenty minutes without breathing, the way you watch a decisive penalty.

A technician's advice: for large files use programs or services that support resuming downloads. That way, if it drops, it starts again from where it got to.

LAW OF THE WEBSITE THAT WORKS EVERYWHERE



The website works on every computer in the office except the one belonging to the client who has to approve it.

— Anonymous, *computing folklore*

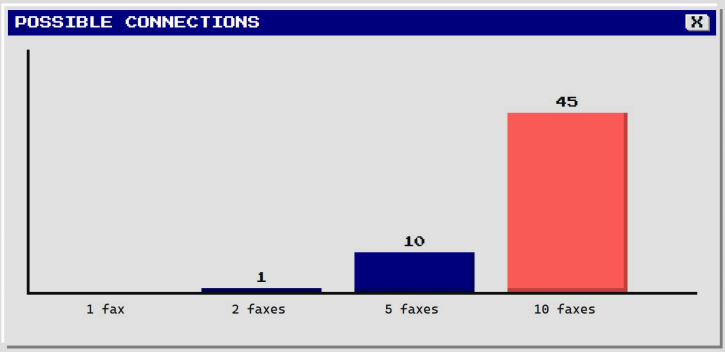
The site has been tested on six browsers, four phones, two tablets and the smart fridge in the staff room. All perfect. Then the client opens it on his computer and sees a menu shoved to the left and a logo the size of a lorry.

The client's computer is a mysterious place. It has a browser that only updates on even-numbered days, six toolbars installed by a nephew in 2016 and the zoom set to a hundred and fifty per cent ‘because it's easier to read’.

And the client never rings to say ‘it doesn't display properly on my computer’. He says: ‘The website doesn't work.’ In the tone of someone announcing an eclipse.

Real advice: before blaming the website, try clearing the cache and setting the zoom back to a hundred per cent. It solves more problems than you'd think.

METCALFE'S LAW



One fax alone is a noisy ornament. Ten faxes are a society, with gossip.

The value of a network grows with the square of the number of connected devices (or people): every new node increases the possible connections of all the others.

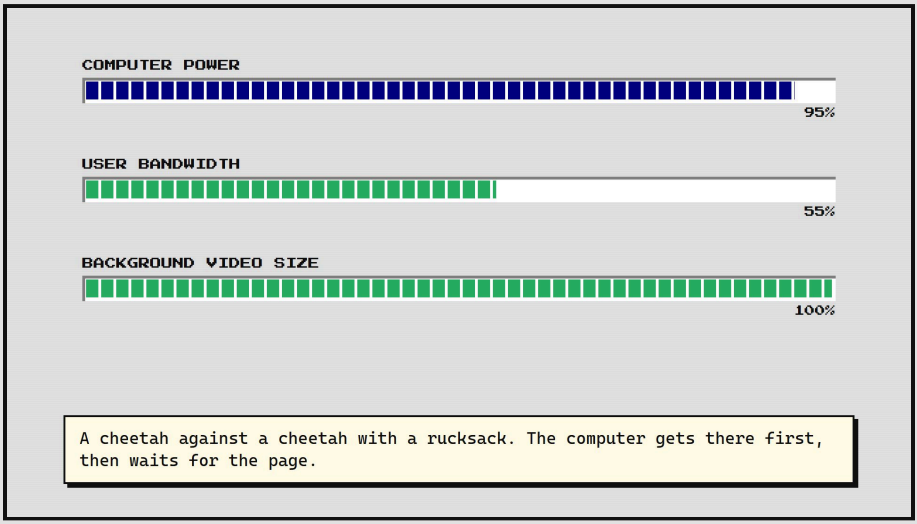
— Robert Metcalfe, *Formulated around 1980 to promote Ethernet; the name 'Metcalfe's Law' was coined by George Gilder, 1993*

A single fax machine is useless. It's an ornament that makes noise. Two fax machines can talk to each other. Ten fax machines already form a society, with gossip and thermal paper. That's Metcalfe's Law: every new member makes the network more valuable for everyone, and not by a little, by the square.

It's why nobody leaves a social network or a group chat: if you go, you lose everyone else. The value isn't in the program, it's in the people trapped inside. Happily trapped, of course.

I know the flip side well. The residents' group chat for our block of flats has forty members. According to the formula it's worth a fortune. In practice it generates sixteen hundred possible rows about where the bikes are parked. That too, mathematically, is value.

NIELSEN'S LAW OF BANDWIDTH



A high-end user's connection speed grows by about 50% a year: more slowly than computing power.

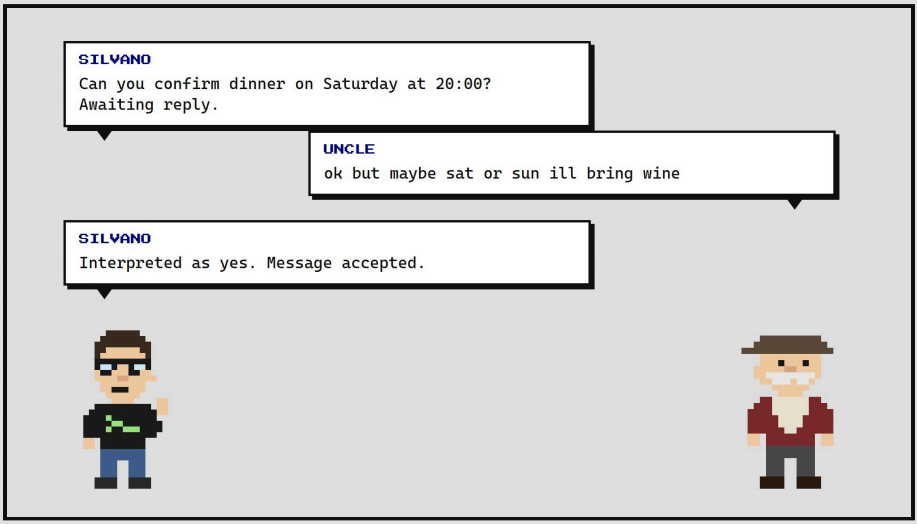
— Jakob Nielsen, *Alertbox, Nielsen Norman Group, 1998 (updated in 2008 and 2019)*

Processors double every two years, bandwidth grows by fifty per cent a year. That sounds like a lot, but it's a race between a cheetah and a cheetah carrying a rucksack. The computer always gets there first. Then it sits down and waits for the page to load.

The lesson for web developers is simple: don't assume everyone has fibre. Somewhere there's always a user in a farmhouse, with a mobile dongle propped on the windowsill at the only spot where it gets a signal, trying to open your ninety-megabyte background video.

I have that client. To improve the signal he put the dongle on top of a bean pole. It works. When it's windy, though, the Internet sways.

THE ROBUSTNESS PRINCIPLE



Be conservative in what you send and liberal in what you accept from others.

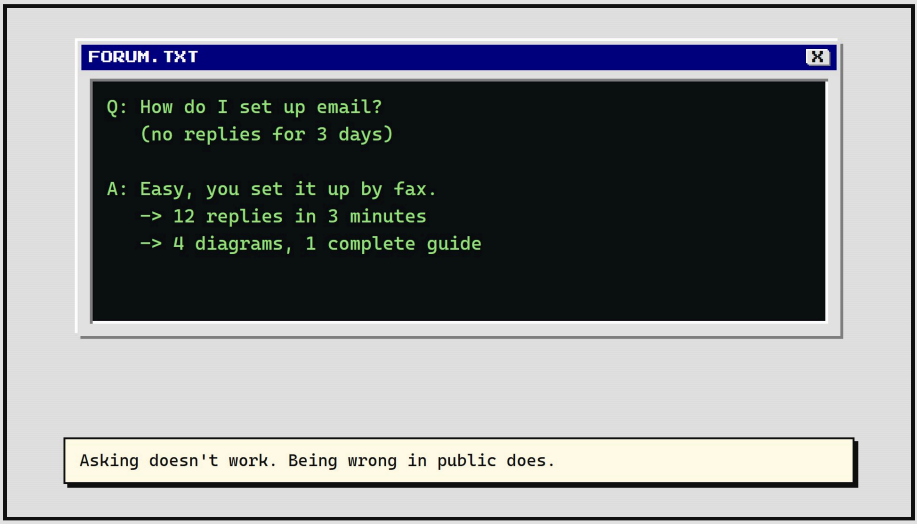
— Jon Postel, *Internet protocol specifications*, RFC 760 (1980) and RFC 793 (1981)

It's the etiquette of the Internet: when you speak, follow the protocol to the letter; when you listen, be understanding with anyone who gets a comma wrong. Thanks to this courtesy, machines written by different people manage to understand each other.

There's a flip side: if everyone accepts errors, nobody ever corrects them. After thirty years of tolerance, browsers read HTML pages that look as if a cat wrote them on the keyboard, and they have to keep doing so for ever.

I applied the principle to my social life. I send impeccable messages, punctuated like a solicitor's letter. I accept any reply, even just a thumbs-up. My aunt replied with an aubergine. I took it as a 'yes'. Now I'm best man at a wedding.

CUNNINGHAM'S LAW



The best way to get the right answer on the Internet is not to ask a question; it's to post the wrong answer.

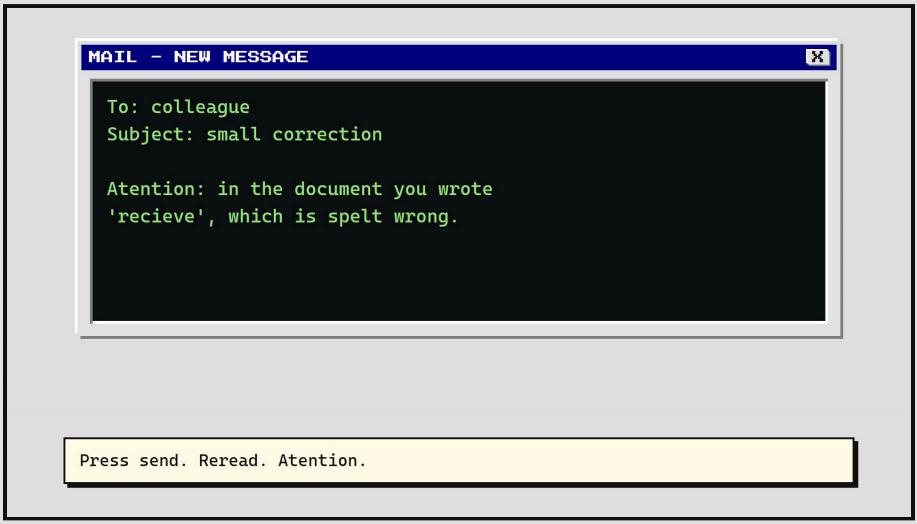
— Steven McGeady, *Named by Steven McGeady in 2010 after Ward Cunningham, who has however denied ever saying it (disputed attribution)*

Ask on a forum how to configure the mail server: silence, a tumbleweed rolls by. Confidently post a wrong procedure: within three minutes twelve experts turn up, with four diagrams and a correction signed by a retired engineer.

It works because correcting someone is more satisfying than helping them. It's a knowledge engine powered by the desire to be right. The final paradox: the man it's named after says he never said it. The law proved itself.

I use it for everything. I wrote in a group that Joomla is installed by fax. Now I have a complete, up-to-date guide with screenshots. And also a fax, which a user sent me to prove it doesn't work.

SKITT'S LAW



Any post correcting someone else's error will almost certainly contain at least one error of its own.

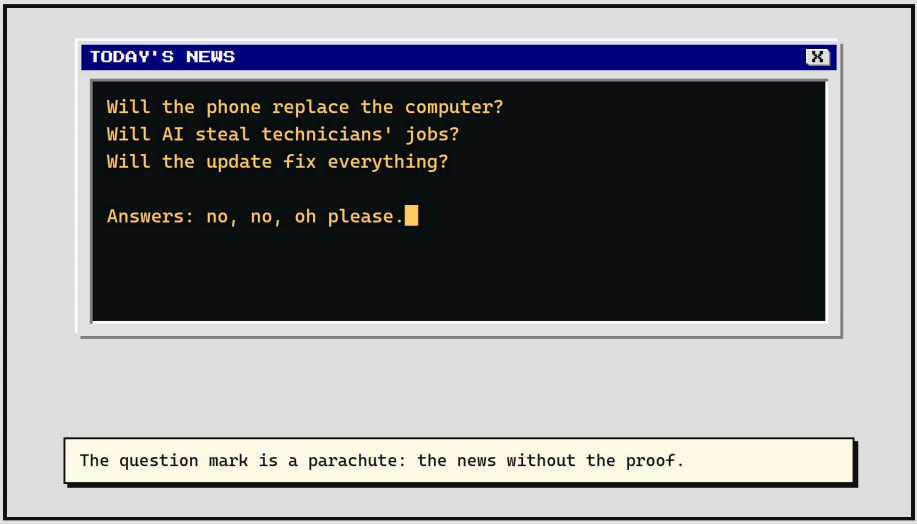
— Anonymous (pseudonym 'Skitt'), *Usenet, alt.usage.english newsgroup, around 1999 (uncertain attribution; earlier variants known as 'Muphry's Law')*

You email a colleague: 'Careful, in the document you wrote "it's" when you meant "its".' You press send. You reread it. You wrote 'Carefull'. The colleague replies with a single word, and that word is perfect.

Skitt's Law is a law of nature of correction. The universe does not tolerate spelling arrogance: every time you play the schoolteacher, a rebel letter detaches itself from the keyboard and hides in the most visible spot in your message. It applies to code too: the patch that fixes your colleague's bug introduces one of your own.

To be safe, I no longer correct anyone. I've delegated it to a spellchecker. Yesterday it changed 'server' to 'severe'. The client read 'the severe is down' and asked whether it was the weather.

BETTERIDGE'S LAW OF HEADLINES



Any headline that ends in a question mark can be answered by the word 'no'.

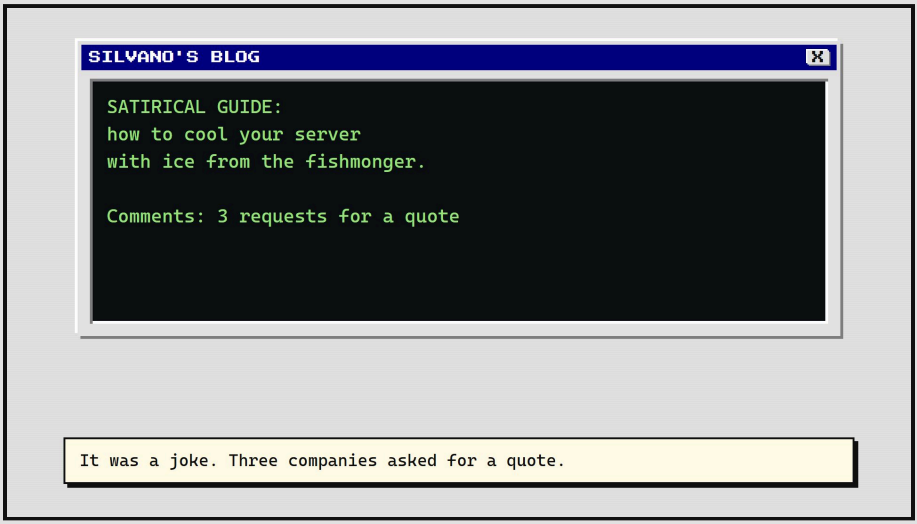
— Ian Betteridge, *Post on his blog, February 2009 (the idea had circulated under other names since the 1980s)*

'Will this new phone replace the computer?' No. 'Will artificial intelligence steal technicians' jobs?' No; if anything, we're the ones who have to configure it. 'Will the new update fix every problem?' Oh, please.

Betteridge's law exposes an ancient trick: if the journalist had evidence for a yes, they'd write a statement. The question mark is a parachute. It lets you launch the sensational story without having to prove it, and pocket the clicks of the hopeful.

I ran an experiment on my blog. I published 'Will the floppy disk come back into fashion?' Three thousand visits. Then I published 'The floppy disk will not come back into fashion'. Two visits, and one was my mother. The question mark, I realised, is the real engine of digital publishing.

POE'S LAW



Without a clear indication of the author's intent, a parody of an extreme view is indistinguishable from a sincere expression of that view.

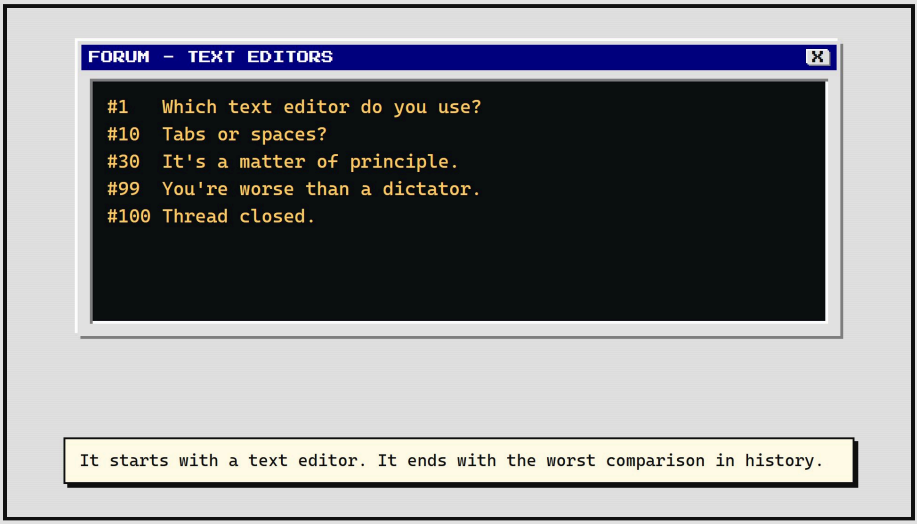
— Nathan Poe, *christianforums.com, 2005 (a similar warning dates back to Usenet, 1983)*

You write online that the best way to protect your password is to print it and stick it on the monitor, so thieves won't look where it's obvious. It's a joke. Three people thank you. One actually does it. Another starts a paid course.

That's Poe's Law: online, irony has no tone of voice, and some real opinions are so extreme that parody can no longer outdo them. That's why emoticons were invented. They are our digital raised eyebrow.

I once published a satirical guide to cooling the server with ice from the fishmonger's. Three companies wrote asking for a quote. One has already ordered the sea bream. He says the logs now smell of the sea.

GODWIN'S LAW



As an online discussion grows longer, the probability of someone making a comparison involving Hitler or the Nazis approaches certainty.

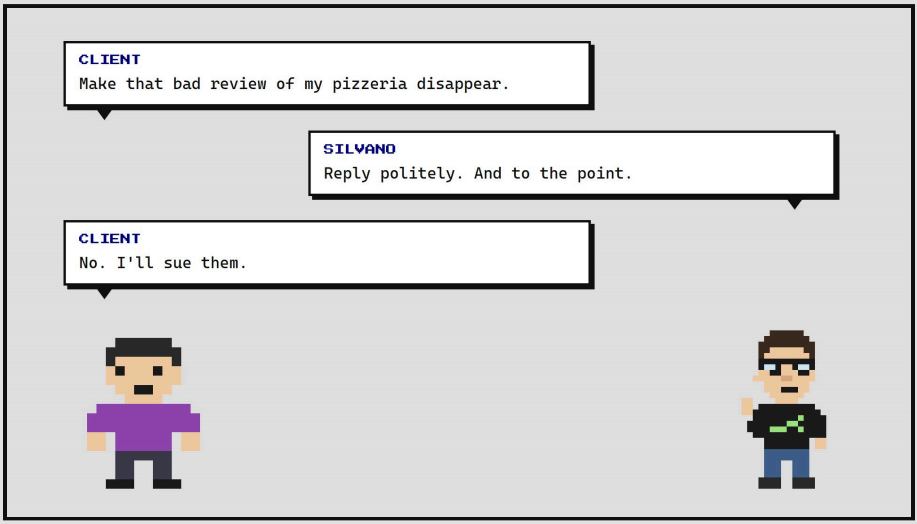
— Mike Godwin, *Usenet, 1990*

Every online discussion starts politely. Someone asks which text editor to use. By the tenth message it's tabs versus spaces. By the thirtieth, moral principles. By the hundredth someone compares their opponent to the worst dictator in history. Over a text editor.

Godwin's Law doesn't say it happens, it says it's inevitable, like gravity or the HDMI cable always being on the wrong side. Old forums also had an unwritten rule: whoever makes that comparison first has lost the argument.

I moderate a forum for printer enthusiasts. We held out for three hundred and twenty messages. Then someone described a certain brand's driver as a totalitarian regime. To be fair, he had a point. But the thread was closed anyway.

THE STREISAND EFFECT



Trying to hide, remove or censor a piece of information online ends up making it far more visible and widespread.

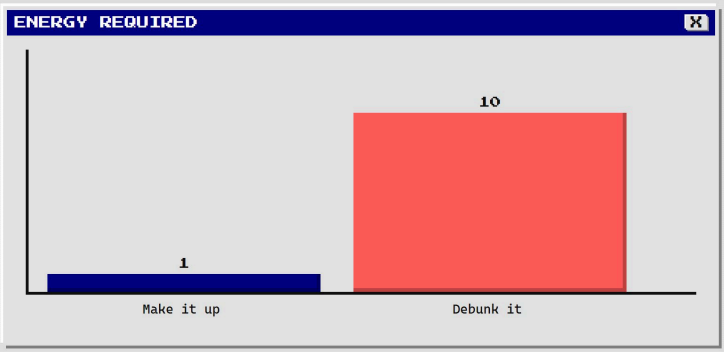
— Mike Masnick, *Techdirt, January 2005 (inspired by a 2003 case involving an aerial photograph)*

The name comes from a 2003 case: a lawsuit to have an aerial photograph of a villa, taken to document coastal erosion, removed from an online archive. Before the lawsuit hardly anyone had seen the photo. Afterwards, half the world had. A blogger built the name of a phenomenon on it.

The Internet works like a cat: tell it not to look in a box and it will look at nothing else. Every removal request is a neon sign saying 'something interesting here'. Copies, archives and screenshots multiply faster than any lawyer.

A client asked me to make a negative review of his pizzeria disappear. I advised him to reply politely. He preferred to threaten legal action. Now the review is the most-read in the county. The pizza, meanwhile, has improved. But nobody knows.

BRANDOLINI'S LAW



Three seconds to make it up. Two hours, four links and a chart to debunk it.

The amount of energy needed to refute nonsense is an order of magnitude bigger than that needed to produce it.

— Alberto Brandolini, *Twitter post, January 2013 (also known as the bullshit asymmetry principle)*

Writing '5G makes geraniums wilt' takes three seconds. Explaining why it isn't true takes two hours, four links, a chart and the patience of a monk. Meanwhile the geranium claim has done the rounds of three family group chats.

The law, formulated by an Italian programmer, describes a cruel asymmetry: making things up is cheap, checking is expensive. It's why hoaxes win every speed race, and why every sysadmin has spent at least one dinner explaining that no, the computer did not catch the virus from the plug socket.

I've found a countermeasure. When a client forwards me a hoax, I reply with an opposite hoax, just as short. They cancel each other out. I've already earned a reputation as a well-informed person. I also have a few geranium-related complaints against me, but they're being looked into.

THE SYSADMIN'S ADVICE

When ‘the Internet is slow’, before arguing with your provider, do the cable test. Connect a computer directly to the router with a network cable and measure the speed with an online test. Then repeat the test over Wi-Fi, in the room where you usually work.

- If **it's slow even with the cable**, the problem is the line or the router: call your provider, with the numbers in hand.
- If **it's fast with the cable but not over Wi-Fi**, the line is innocent. The problem is the signal at home: router position, walls, interference.

Two useful details. Modern routers broadcast on two bands: 5 GHz is faster but gets through walls less well, 2.4 GHz reaches further but is more crowded. And in large spaces, a mesh system or an access point connected by cable works far better than cheap repeaters. Sometimes moving the router two metres is worth as much as changing contract.

CHAPTER 5

CH 05 — DEBUG

Bug hunting

Now we enter the lab. A bug, for those who don't keep company with them, is an error in a program. The word is older than computers, but there is a 1947 logbook with a real moth taped to it, found inside a machine. The hunt has never stopped since, and the moth has had a great many grandchildren.

The laws in this chapter show the bug in its natural habitat. It vanishes when a colleague arrives, hides in the last line you check, waits for Friday afternoon, turns up ten minutes after release. Sometimes, if it survives long enough, users grow fond of it and it becomes a feature.

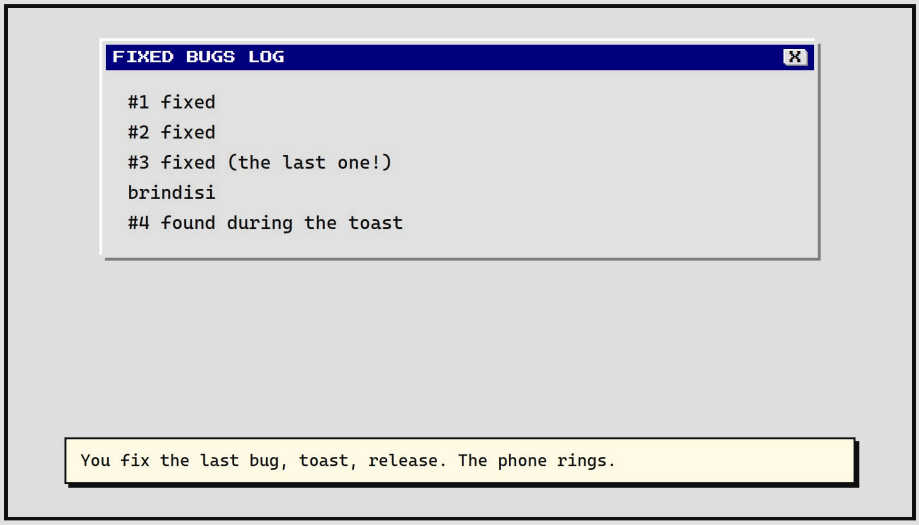
Non-programmers will find here the answer to the question they have been asking for years: why does the program I use every day sometimes do strange things? Short answer: because there is always one more bug. Long answer: this chapter.

Programmers, on the other hand, will recognise themselves on every page. I apologise in advance for the nights I will bring back to mind.

CONTENTS.DIR

01 Lubarsky's Law of Cybernetic Entomology
02 Law of 'it works on my machine'
03 Law of the miracle-working colleague
04 Law of the innocent line
05 Kernighan's Law
06 Law of the prolific fix
07 Law of the evolved bug
08 Law of the punctual bug
09 Law of Friday afternoon
10 Law of the untested part
11 Law of 'it compiles, so it works'
12 Law of the test data
13 Law of the oracular message
14 Law of the treacherous calendar
15 Linus's Law
16 Hyrum's Law

LUBARSKY'S LAW OF CYBERNETIC ENTOMOLOGY



There's always one more bug.

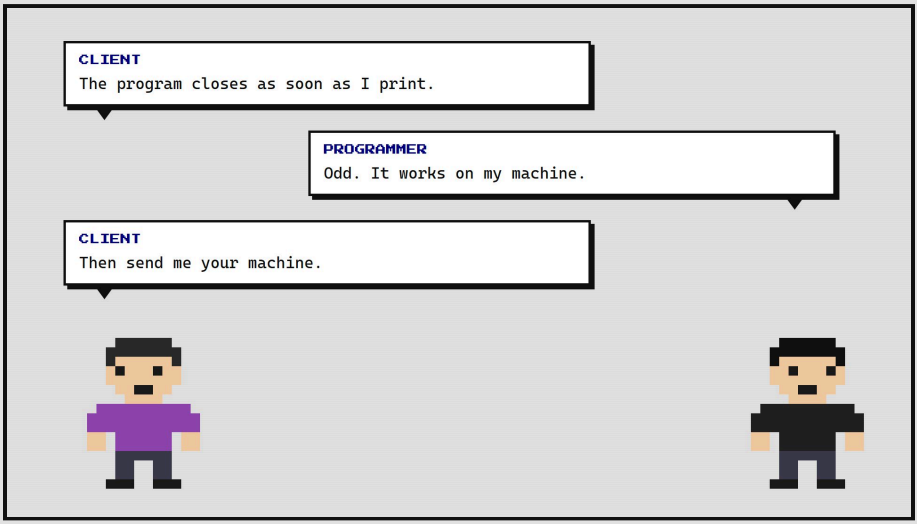
— Anonymous, *Saying circulated in collections of computer humour (e.g. Multics and Usenet lists); identity of Lubarsky unverified (uncertain attribution)*

Five words and every programmer's career is summed up. You fix the last bug, raise a glass, release. The phone rings. There was a last-last bug.

The meaning is wise humility: no non-trivial program is free of errors. You can make them rare, you can make them harmless, but thinking you've caught them all is the most dangerous bug of all.

I called pest control for the server. He sprayed something on the case, said 'six months' guarantee' and left. The next day the management system was printing invoices in Cyrillic. I rang him back. He told me that species wasn't covered. Cybernetic entomology: the exact science of always.

LAW OF 'IT WORKS ON MY MACHINE'



Any error reported by the client is impossible to reproduce on the programmer's computer, where everything works perfectly.

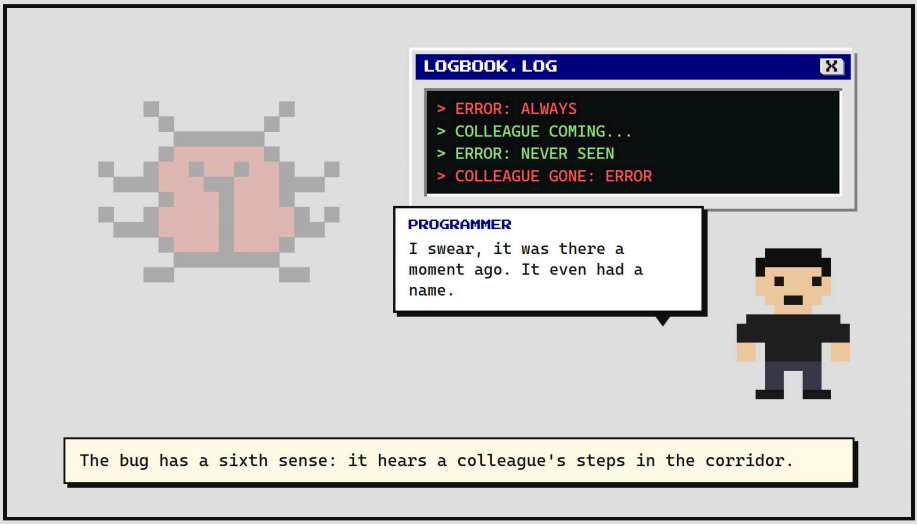
— Anonymous, *computing folklore*

It is the most spoken sentence in the history of computing, more than 'have you tried turning it off and on again'. The programmer says it with touching sincerity: on their computer the program really does work. The trouble is that the client doesn't live inside their computer.

The developer's computer is a protected ecosystem, a tropical greenhouse with all the right libraries, the right settings, the right language. The client's computer is a jungle with Windows from 2016, three antivirus programs at war with each other and the clock set to Honolulu time.

There was a period when I seriously considered shipping my PC to clients along with the software. It would have solved ninety per cent of the problems. Then I realised the client would ring anyway, to say that my PC on their desk didn't work.

LAW OF THE MIRACLE-WORKING COLLEAGUE



The bug that has been tormenting you for hours vanishes the instant you call a colleague over to look at it, and reappears as soon as they go back to their desk.

— Anonymous, *computing folklore*

Every office has a colleague with powers. They can't program any better than you, they didn't study any longer. Simply, when they approach your monitor, the bugs stand to attention and pretend not to exist.

'Show me,' they say. You repeat exactly the same steps, in the shaky voice of someone reporting a ghost. The program works. They pat you on the shoulder with the air of a family doctor. They leave. The bug reappears, and smiles at you.

Programmers even have a name for these shy errors: bugs that change behaviour when you observe them. I solved it at the root. I printed a life-size photo of the colleague and hung it behind my chair. It works sixty per cent of the time.

LAW OF THE INNOCENT LINE

SUSPECTS, IN ORDER

1. my colleague's code
2. the external library
3. the operating system
4. cosmic rays
5. my line (impossible)

Six hours of investigation. The culprit was the chairman of the inquiry.

The bug is always in the last line you check, and it is the one you were absolutely sure was correct.

— Anonymous, *computing folklore*

Bug hunting follows a precise ritual. First you suspect your colleagues' code, then the external library, then the operating system, then the compiler, and finally cosmic rays. Only at the very end, reluctantly, do you look at your own line. The perfect one. The one written with pride.

And there it is, the error. A misplaced semicolon, a single equals sign instead of a double. Six hours of investigation for one character. An entire public inquiry to discover that the culprit was the chairman of the committee.

There's a logical explanation too, which ruins the poetry: you stop looking when you find it, so it's always in the last place you look. But there's no point telling a programmer that at two in the morning. They know the code is mocking them. And, honestly, they're right.

KERNIGHAN'S LAW

TUE 02:10

BRILLIANT IDEA

At two a.m. you write one line that does twelve things. You feel like a galactic genius.

Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.

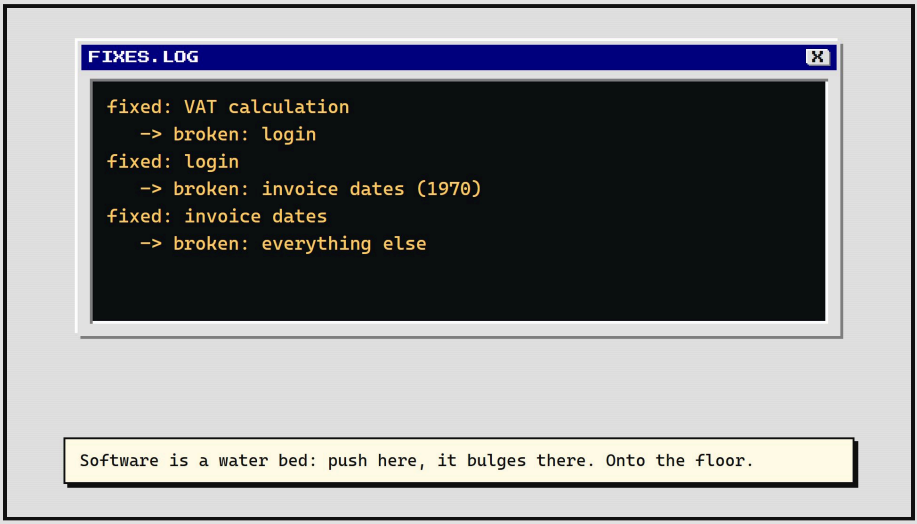
— Brian Kernighan, *The Elements of Programming Style (with P. J. Plauger), 2nd ed., 1978*

It's a ruthless syllogism. At two in the morning, inspired, you write a line that does twelve things at once. You feel like a galactic genius. The next morning the line doesn't work and the night-time genius isn't answering the phone.

The real message is: write simply. Clever code is a debt owed to your future self, who will be more tired, older and short of coffee.

I've solved it methodically: I write code at 50% of my intelligence, so I can debug it at 100%. My colleagues say it shows. I take it as a compliment to the engineering, not to me.

LAW OF THE PROLIFIC FIX



Every bug fixed in a hurry generates at least two new ones, preferably in a part of the program that had nothing to do with it.

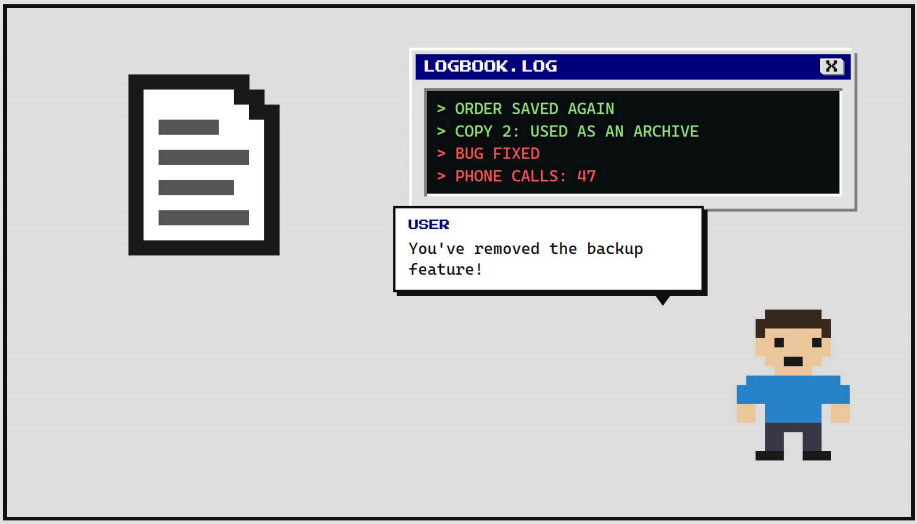
— Anonymous, *computing folklore*

Bugs are like the heads of the hydra in the legends: cut one off and two grow back. With one difference: the hydra's heads at least grew back in the same place. New bugs prefer to appear on the other side of the program, where nobody is looking.

You fix the VAT calculation and the login stops working. You sort out the login and the invoices come out dated 1970. Software is a system of communicating vessels: push on one side and it comes out of the other, usually on the floor.

At this point the quality guru utters the magic words: 'automated tests'. He's right. But the urgent bug always arrives when the automated tests are 'nearly ready'. I approach the quick fix in the same spirit as changing a light bulb in an old house: with confidence, and with the fire extinguisher close by.

LAW OF THE EVOLVED BUG



Any sufficiently advanced bug is indistinguishable from a feature.

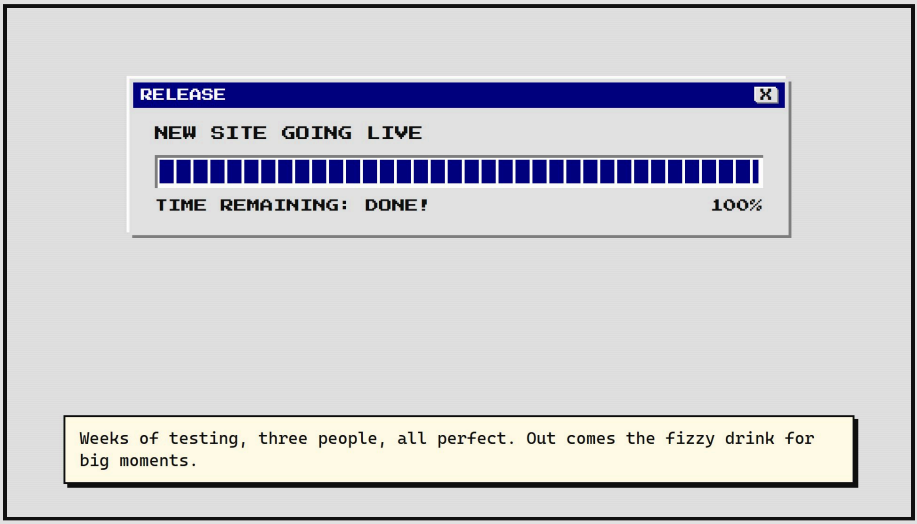
— Anonymous, *computing folklore*

Every so often an error survives long enough to put down roots. Users discover it, get used to it, find it handy. At that point fixing it becomes an act of violence.

A true example, told with due respect: a management system that, by mistake, saved draft orders twice. The staff had learnt to use the second copy as an archive. When we fixed it, they rang en masse: 'You've removed the backup feature.'

Hence the famous phrase every programmer utters at least once in their life, chin up and gaze fixed on the horizon: 'It's not a bug, it's a feature.' It is a magic formula, like abracadabra. It solves nothing, but it turns shame into marketing. And marketing, as everyone knows, cannot be fixed.

LAW OF THE PUNCTUAL BUG



Weeks of testing, three people, all perfect. Out comes the fizzy drink for big moments.

The critical bug is discovered ten minutes after release, never ten minutes before.

— Anonymous, *computing folklore*

For weeks the program is tested, retested, checked by three different people. All fine. Then someone presses the 'publish' button, the fizzy drink of great occasions is opened, and at that precise instant someone says: 'Erm, sorry…'

It isn't bad luck, it's an almost physical phenomenon. While the code is under test, the mind is in defensive mode and sees nothing. As soon as the code is public and irreversible, the mind relaxes and suddenly becomes brilliant. It sees everything. Too late, but it sees everything.

I tried to cheat fate: I pretend to publish, wait ten minutes, then publish for real. It worked twice. The third time I found the bug in the tenth minute of the fake release and celebrated it like a goal. Then came the second bug, the real one.

LAW OF FRIDAY AFTERNOON



A change put into production on Friday afternoon will break after six in the evening, when all your colleagues are already on the motorway.

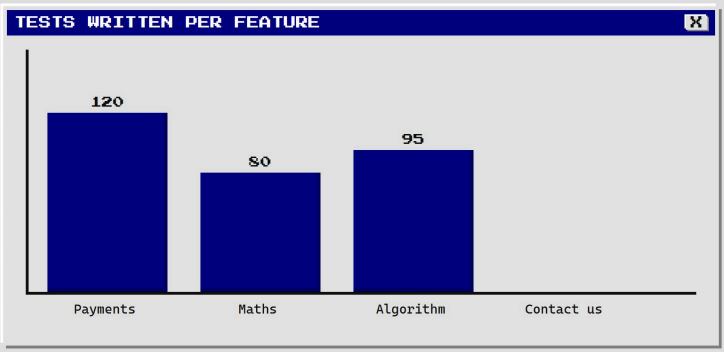
— Anonymous, *computing folklore*

In the IT calendar, Friday afternoon is a sacred and cursed zone, like the borderlands in fantasy sagas. Whoever releases an update in those hours defies the ancient gods of production, and the ancient gods do not forgive.

It always goes the same way. Half past four: 'It's a tiny change, just one line.' Five o'clock: all green, all perfect. Ten past six: the website shows the home page in Chinese and the shopping basket is giving the goods away.

Meanwhile, on the motorway, the colleague who knows the system has their phone in aeroplane mode for spiritual reasons. At our agency there's an unwritten decree: on Fridays we only do reversible things. Dusting the keyboard, reading documentation, thinking. Above all thinking, which never goes into production.

LAW OF THE UNTESTED PART



A fortress around payments. The back door tied shut with string.

The part of the program that breaks in production is always the one nobody thought of testing, because it was too simple to get wrong.

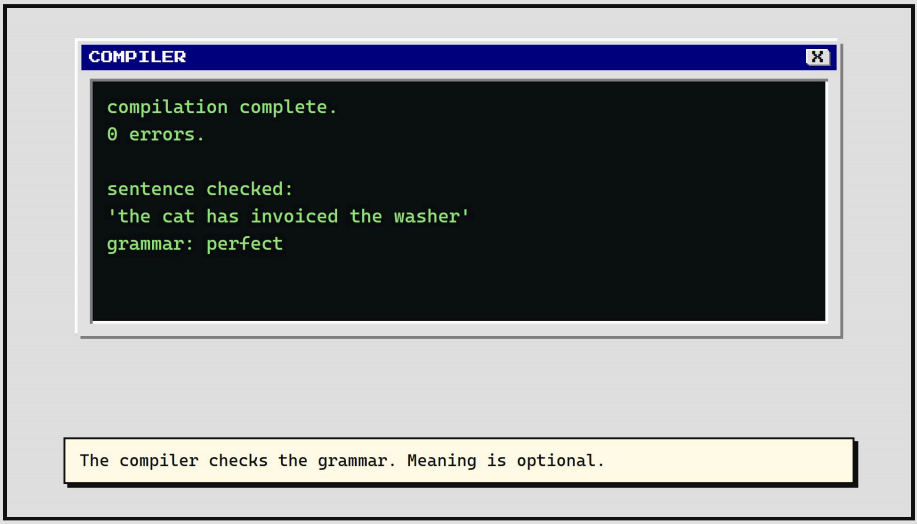
— Anonymous, *computing folklore*

Tests always focus on the difficult parts: complex calculations, algorithms, payments. Fortresses are built around those. Then the website collapses because of the 'Contact us' button, which nobody had ever clicked because 'it's only a form'.

It's the computing version of the medieval castle with walls three metres thick and the back door tied shut with string. The enemy doesn't scale the walls. The enemy, quite simply, knocks round the back.

The high priest of quality has been repeating the same sermon for years: 'Whatever isn't tested is broken, you just don't know it yet.' It's a frightening sentence, because it's true. Now I test the trivial things too, one by one, with the zeal of a solicitor. The client asks why it takes me so long. I tell them I'm checking the back door.

LAW OF 'IT COMPILES, SO IT WORKS'



If the code compiles without errors, the tired programmer will consider it ready for delivery.

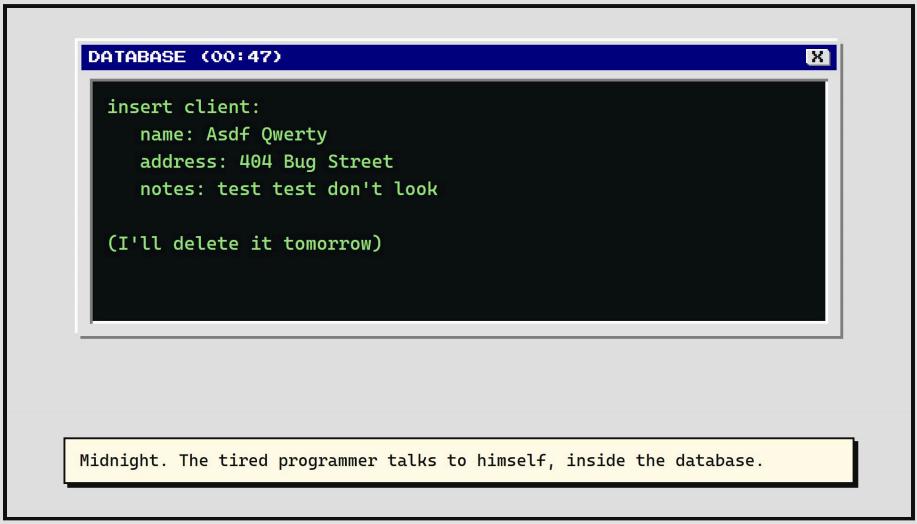
— Anonymous, *computing folklore*

Compiling means translating code into an executable program. The compiler checks that the grammar is correct, not that the program makes sense. It's like a proofreader who enthusiastically approves the sentence 'the cat invoiced the washing machine'.

And yet, at seven in the evening, faced with the words 'build succeeded', something clicks in the human brain. An inner voice says: done. The code sets off for the client like a ship leaving port with its sails on backwards.

In many offices the short version of the motto does the rounds: 'It compiles? Ship it.' People say it laughing. Then they put it into practice. I did it just once, long ago. The washing machine, figuratively speaking, is still spinning. And I've learnt to try the program first, at least once.

LAW OF THE TEST DATA



The most embarrassing test data entered during development will, sooner or later, appear in front of the most important client.

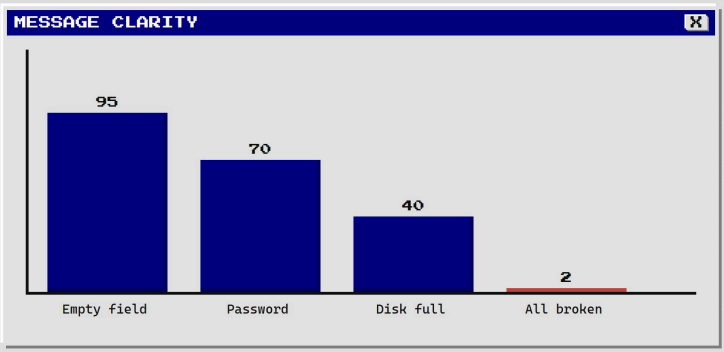
— Anonymous, *computing folklore*

Nobody enters realistic data during development. You type 'asdf', 'test test', 'Joe Bloggs', 'product that doesn't exist'. They are the placeholders of the tired programmer, their way of talking to themselves at midnight.

Then comes the official presentation. Meeting room, projector, managing director in the front row. The customer list opens and at the top, in bold, is Mr Asdf Qwerty, of '404 Bug Street'. Silence. Someone coughs.

The rule I learnt the hard way is simple: write every piece of test data as if the client were going to read it. Because they will. Today my test environments contain only sober, respectable names like 'Example Client Ltd'. A bit sadder, but far less sweat.

LAW OF THE ORACULAR MESSAGE



The worse the error, the less it says. Like an oracle low on credit.

The more serious an error is, the vaguer, more generic and more incomprehensible the message announcing it.

— Anonymous, *computing folklore*

Small errors speak plainly: 'The Email field is not valid.' Catastrophic errors, on the other hand, express themselves like the oracles of antiquity: 'An unexpected error has occurred.' Code 0x8000FFFF. End of communication.

Unexpected by whom, exactly? It is a sentence written by a programmer who, years earlier, had foreseen perfectly well that something would go wrong, but couldn't be bothered to say what. It's a note left on the fridge: 'Something happened. Bye.'

Clients read them to me over the phone in the voice of someone reciting a prophecy. I listen in silence, nod, and then pronounce the sacred formula every sysadmin knows: 'Have you tried turning it off and on again?' It doesn't always work. But it's the only answer worthy of the question.

LAW OF THE TREACHEROUS CALENDAR

DATES THE PROGRAM HATES

- 29 February
- the night the clocks change
- 1 January
- 30 February (its own invention)

Three hundred and sixty-two perfect days. The other three: gothic novel.

Any program that handles dates will work all year round, except on 29 February, on the night the clocks change and on the first day of January.

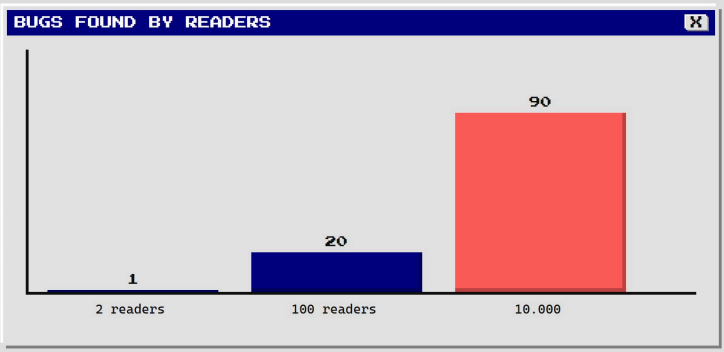
— Anonymous, *computing folklore*

Time seems simple. Sixty seconds, sixty minutes, twenty-four hours. Then a programmer tries to handle it in code and discovers the horror: leap years, time zones, summer time, twenty-eight-day months, countries that change their minds.

The result is software that works beautifully three hundred and sixty-two days a year. On the other three it does things out of a Gothic novel: bookings in the past, payslips with a phantom hour, deadlines set for 30 February.

One night at the end of October, one of my management systems recorded the same clock-in twice, at half past two, because that hour really had existed twice. The employee appeared to be in two places at once. The director wanted to promote him. I fixed the bug, with the regret of someone who has destroyed a miracle.

LINUS'S LAW



Open code and many readers: sooner or later the flaw pops out.

Given enough eyeballs, all bugs are shallow.

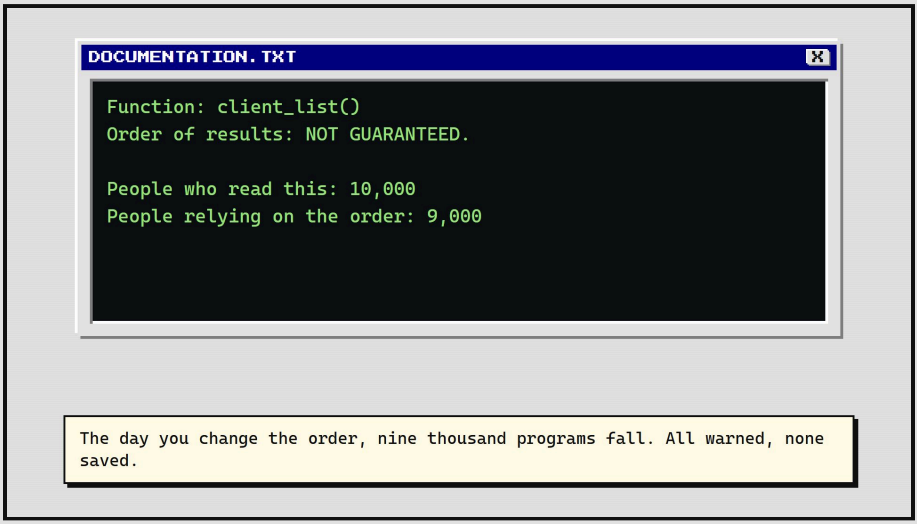
— Eric S. Raymond, *The Cathedral and the Bazaar, 1997 (essay), 1999 (book); named in honour of Linus Torvalds*

The idea is beautiful: if the code is open and thousands of people read it, someone will find the defect. It's the reason free software can compete with big companies, and often win.

The awkward detail is the word 'enough'. The eyes also have to actually look. There are libraries used by half the planet and read, in practice, by two tired volunteers and a cat.

I tried applying it at the office: I projected the code on the wall during the coffee break. Twenty eyes. No bugs found, but three colleagues reported that the font was ugly and one asked whether the projector was new. The bug is still there, beautifully lit.

HYRUM'S LAW



With a sufficient number of users, every observable behaviour of your system will be depended on by somebody, regardless of what you promised in the documentation.

— Hyrum Wright, *Observation born in software engineering at Google, 2010s (hyrumslaw.com)*

You write in the documentation: 'The order of the results is not guaranteed.' Ten thousand people read it. Nine thousand build their program assuming the order is always the same. The day you change it, half the Internet falls over.

The law tells us that the real contract of a piece of software isn't the written one, but everything it does, defects included. Fixing a bug can break someone who had built a business on that bug.

Once I corrected a typo in an error message. The next day a furious client rang me: his script looked for exactly that misspelt word to tell that something had gone wrong. I restored the typo. It's now a listed monument.

THE SYSADMIN'S ADVICE

Bug hunting is not fortune-telling, it's a method. Mine, tested over quite a few nights, goes like this.

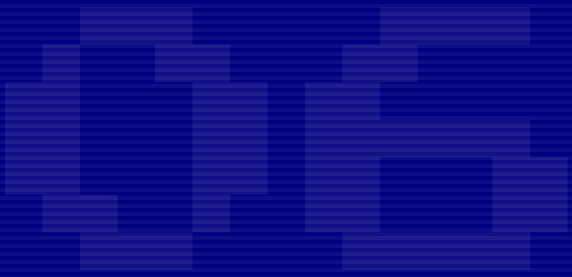
- **Reproduce it first.** Write down the exact steps that make the error appear. If you can't make a bug come back, you won't be able to tell whether you've fixed it.
- **Read the logs.** The program has often already written down somewhere what went wrong. It sounds obvious: it is the most often skipped step.
- **Split in half.** If it worked yesterday and not today, look halfway for what changed. Git even has a command for it, git bisect, which finds the guilty change by halving the field each time.
- **Explain it out loud.** To a colleague, or to an object on the desk. As you tell the problem line by line, the wrong line often gives itself away.

And when you find it, write a test that catches it again if it comes back. Bugs always come home, like cats.

NOTICE

This page has been intentionally left blank.

This is a Murphy's law too: the blank page always turns up where it isn't needed.



CHAPTER 6

CH 06 — CODE

The art of writing code (and regretting it)

If the previous chapter was the crime pages, this one is the philosophy treatise. The laws that follow are not about single errors but about how software gets written: how things are named, how much to comment, when to optimise, why today's clever code is tomorrow's headache.

Many bear the signature of great names in programming, people who built languages, systems and books that generations have studied. It is reassuring to find out that they too, at some point, faced a program they had written themselves a few months earlier and understood nothing of it. Genius is no vaccine.

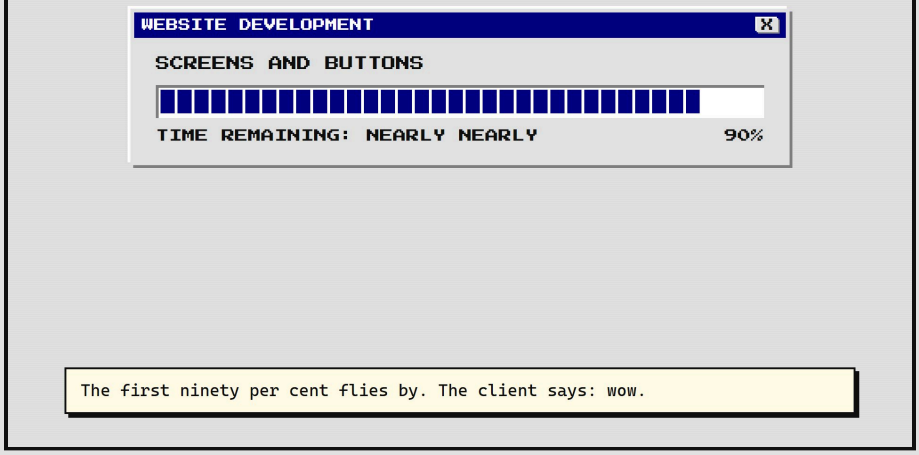
There is only one common thread, and I'll say it now so those in a hurry can skip to the end: simplicity is hard, but it is the only thing that lasts. The rest is commentary. Lying, probably.

For non-programmers: no fear. Every law is explained with everyday examples, and it will help you understand why your management software takes two years to move a button. It isn't laziness. It's archaeology.

CONTENTS.DIR

01 The ninety-ninety rule
02 Law of the two hard things
03 Law of the two problems
04 Eagleson's Law
05 Law of the lying comment
06 Law of posthumous documentation
07 Law of adopted code
08 Osborn's Law
09 Flon's Axiom
10 Hoare's principle
11 Premature optimisation
12 Pike's rules
13 The Law of Demeter
14 The law of leaky abstractions
15 Greenspun's tenth rule
16 Wadler's Law
17 Lehman's Laws

THE NINETY-NINETY RULE



The first 90% of the code accounts for the first 90% of the development time. The remaining 10% accounts for the other 90%.

— Tom Cargill, *Bell Labs; made famous by Jon Bentley in the Programming Pearls column, Communications of the ACM, 1985*

Maths says 180%. The programmer says: exactly. The project is always 'nearly finished' for a period that lasts as long as the whole project.

The first 90% is the fun part: screens, buttons, the client saying 'wow'. The last 10% is when you discover that a surname can contain an apostrophe, that someone was born on 29 February and that VAT changes the very week of release.

In my office there's a website that has been at 98% since 2017. Every year we celebrate the birthday of 'nearly there'. We have a cake. The candles are at 98%: half lit, but with great enthusiasm.

LAW OF THE TWO HARD THINGS

PRICELIST.PHP

```
price = cache.read('coffee')
> 1.00 euro
(the new price list says 1.20)
> clear cache
> 1.00 euro
(the cache is attached to it)
```

The cache remembers everything. Especially what it should forget.

There are only two really hard problems in computer science: cache invalidation and naming things.

— Phil Karlton, *maxim attributed to Karlton in his Netscape days, reported by Tim Bray and Martin Fowler ('TwoHardThings')*

People who don't program think the hard parts are maths, security, artificial intelligence. Wrong. The two real mountains are these: making the computer forget an old thing, and deciding what to call a new one.

The cache is the fast memory that keeps data aside so it doesn't have to fetch it every time. It works brilliantly, until it proudly shows you yesterday's price. Naming, on the other hand, is philosophical torture: 'data', 'data2', 'newData', 'newDataReally'. I've seen whole teams stuck for an afternoon on a variable name, like a medieval church council.

There is also a folk variant that adds a third problem: off-by-one errors in counting. There are two hard things, but nobody has ever managed to count them right.

LAW OF THE TWO PROBLEMS

EMAIL_CHECK

```
rule = /^[a-z.]+@+[a-z]*\.it$/

test('foo@@.it')           -> VALID
test('office@accountant.com')
                           -> REJECTED
```

A formula to recognise emails. Accepts foo, rejects the accountant.

Some people, when confronted with a problem, think: 'I know, I'll use regular expressions.' Now they have two problems.

— Jamie Zawinski, *Usenet post from 1997, traced by Jeffrey Friedl on the regex.info blog (2006)*

Regular expressions are formulas for searching text: extremely powerful and written in a language that looks like a cat walked across the keyboard. They are used, for example, to check whether an email address is valid. The result is a line of symbols that no human being can read after ten minutes, including the person who wrote it.

The first problem was finding the emails. The second is working out why the formula accepts 'bob@@.co.uk' and rejects the accountant. The third, because there's always one more, is that the only colleague who understood it has changed jobs.

I use them with the caution you'd show a spell found in the attic. They work, but every time I change one I seem to hear thunder in the distance. And usually, in fact, the server goes down.

EAGLESON'S LAW

OLD_PROJECT.PHP

```
function f($foo2) {
  $x = $foo2 * 3.14 + 7;     // ?
  // fix later
  return f2($x, true, true, false);
}
```

No comments, mysterious variables. Who did this?

Any code of your own that you haven't looked at for six or more months might as well have been written by someone else.

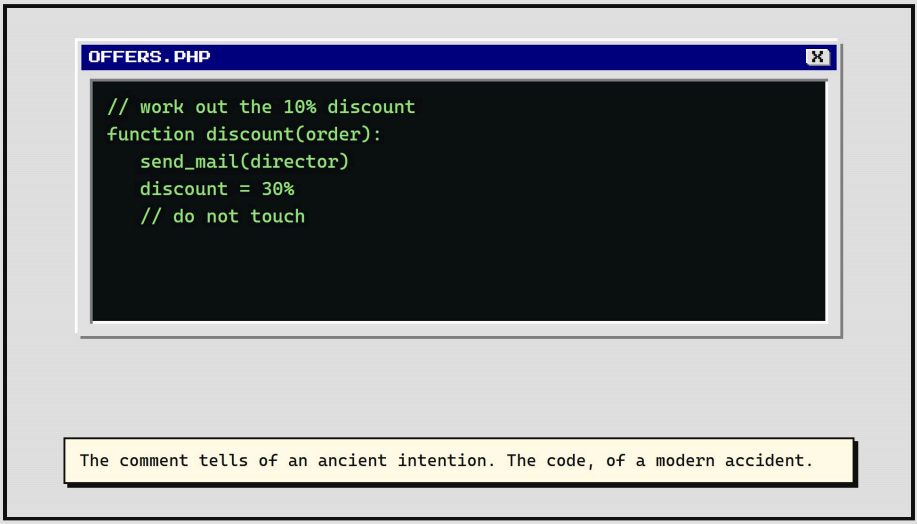
— Anonymous, *Saying common among programmers; origin and identity of Eagleson unverified (uncertain attribution)*

You open an old file. Who wrote this thing? Variables called 'foo2', no comments, a three-hundred-line function. You check the author. It's you. Yourself from six months ago, a total stranger with your login.

The law teaches that the first reader of the documentation isn't your colleague: it's you in the future, who will have forgotten everything. Writing clear code is an act of kindness towards a stranger who looks a lot like you.

These days I treat my past self as an outside consultant. I send him complaint emails. He never replies. Yesterday I found one of his comments: 'Fix later.' Later when? We're in later now. Later is me.

LAW OF THE LYING COMMENT



When the comment and the code disagree, both are probably wrong.

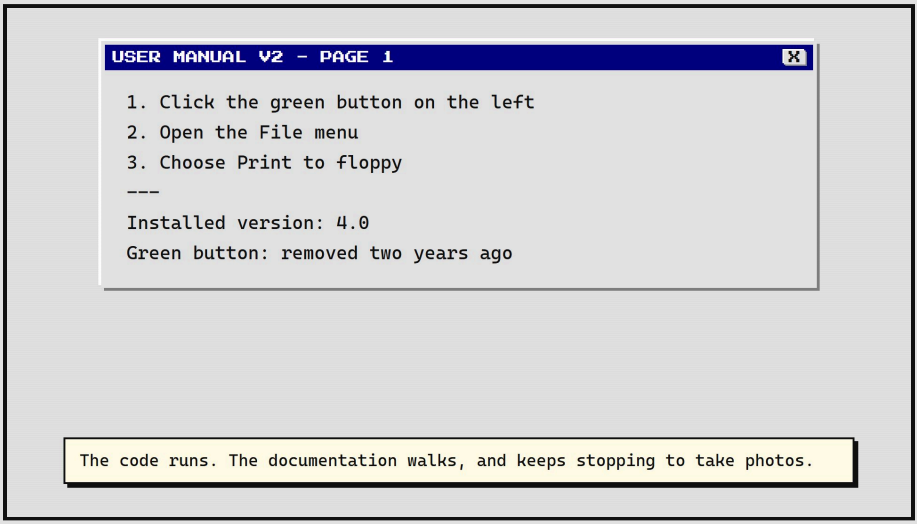
— Anonymous, *computing folklore*

Comments are the sentences programmers leave in the code to explain what it does. In theory they are an act of kindness towards whoever comes next. In practice they are like road signs for a road that has since been moved.

The code gets changed, the comment stays. So you find 'calculates the ten per cent discount' above a function that sends an email to the managing director. Who's right? Nobody. The comment tells of an ancient intention, the code of a modern accident.

There is also the opposite school of thought, the proud programmers who never comment: if it was hard to write, it should be hard to understand. I'm somewhere in the middle. I always comment, but with phrases like 'do not touch'. It explains nothing, but it saves a lot of lives.

LAW OF POSTHUMOUS DOCUMENTATION



Documentation, when it exists, describes the previous version of the program with great precision.

— Anonymous, *computing folklore*

Technical documentation is a literary genre of its own, halfway between the historical novel and the last will and testament. It describes in loving detail a world that no longer exists: buttons that have moved, menus that have been renamed, features removed in the meantime during a four-minute meeting.

The reason is simple. Code runs, documentation walks. By the time someone finally writes the manual for version two, the programmers are already arguing about version four.

I have a client who keeps the manual for his management software in a display case, like a relic. Every now and then he opens it and reads the instructions aloud, solemnly. None of them matches the screen any more. But he says it helps him remember what the world was like when the buttons were on the left. And frankly, I understand him.

LAW OF ADOPTED CODE

FORUM_SOLUTION

```
// found on a forum, year unknown
// no idea why, but needed
x = x + 0
wait(0 seconds)
repeat 1 time:
   work()
```

Lines donated by a stranger. Accepted with gratitude and no questions.

Code copied from the Internet works perfectly until you try to understand why it works.

— Anonymous, *computing folklore*

Every programmer has a secret archive of lines borrowed from forums: solutions written by strangers, in another era, for another problem. You paste them in, they work, and you look at them with the gratitude of someone receiving a kidney from an anonymous donor.

Trouble comes when, out of conscientiousness, you try to understand them. You read, you reason, you remove a line that looks useless. Everything collapses. You put the line back. Now it doesn't work even with the line. The adopted code is offended and has decided not to forgive you.

Since then I've treated these lines the way the ancients treated amulets: respect, distance, no questions. My code still contains pieces with the comment 'no idea why, but it's needed'. They are the most stable lines in the whole project. Knowledge is sometimes overrated.

OSBORN'S LAW

CONFIG.PHP

```
const CURRENT_YEAR = 2019;
const VAT_RATE = 20;       // then 21, then 22
var temperature = 0;       // never changed

echo CURRENT_YEAR;
> 2019
```

The variable hasn't moved since day one. The constant changed its mind several times.

Variables won't; constants aren't.

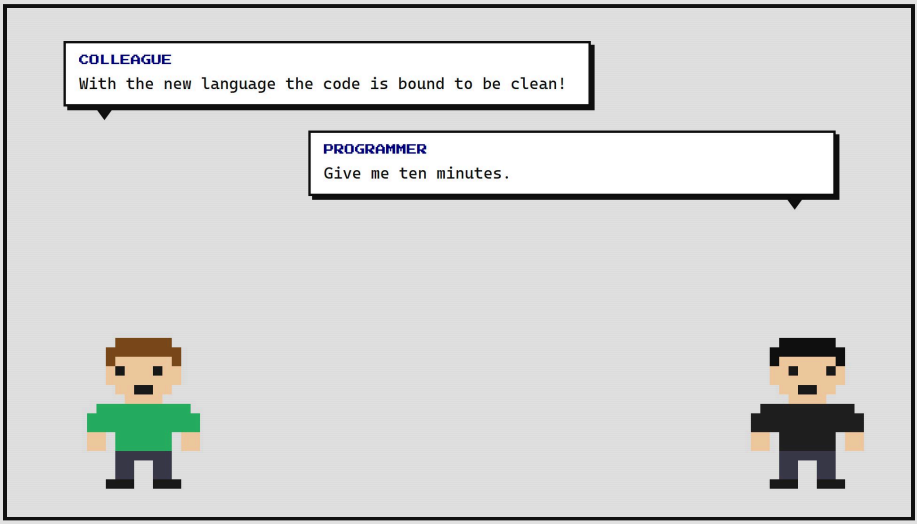
— Anonymous, *Saying common in computer humour; origin and identity of Osborn unverified (uncertain attribution)*

In theory the variable changes and the constant stays put. In practice the variable has been stuck on the wrong value since the day it was created, while the constant 'VAT rate' has changed four times in ten years.

Behind the joke there's a lesson: the things you take as unchangeable, like taxes, dialling codes and date formats, are the ones that will change. And the ones that ought to update often stay frozen because of an initialisation error.

My code contains a constant called CURRENT_YEAR. Its value is 2019. Nobody has the courage to touch it. We've decided that, as far as the software is concerned, it's still 2019. It's nice there. Prices are lower.

FLON'S AXIOM



There does not now, nor will there ever, exist a programming language in which it is the least bit hard to write bad programs.

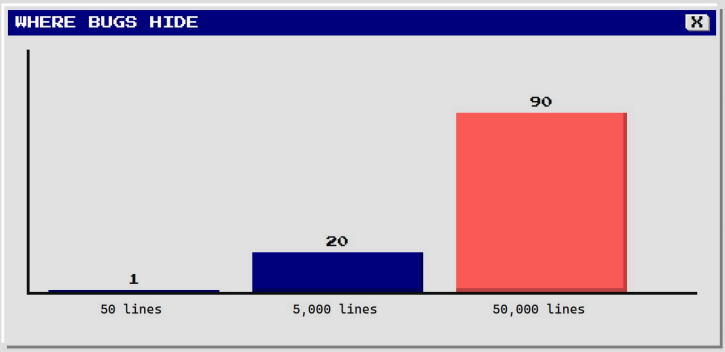
— Lawrence Flon, *On Research in Structured Programming*, *SIGPLAN Notices*, *October 1975*

Every new language arrives with a messianic promise: 'With me, code will finally be clean!' Three months later someone has already written a nine-hundred-line function called 'handle_everything'.

Flon wasn't saying all languages are the same. He was saying that quality lies with the writer, not the tool. A good language helps, but it doesn't stop you building a labyrinth.

I went on a course run by a high priest of modern languages. He showed us perfect code: elegant, safe. Then he left us alone for ten minutes. When he came back we had reinvented global variables, copy-and-paste and a folder called 'final_really'. The language was innocent. We were not.

HOARE'S PRINCIPLE



In fifty lines a bug is a fly on the tablecloth. In fifty thousand it has a house and garden.

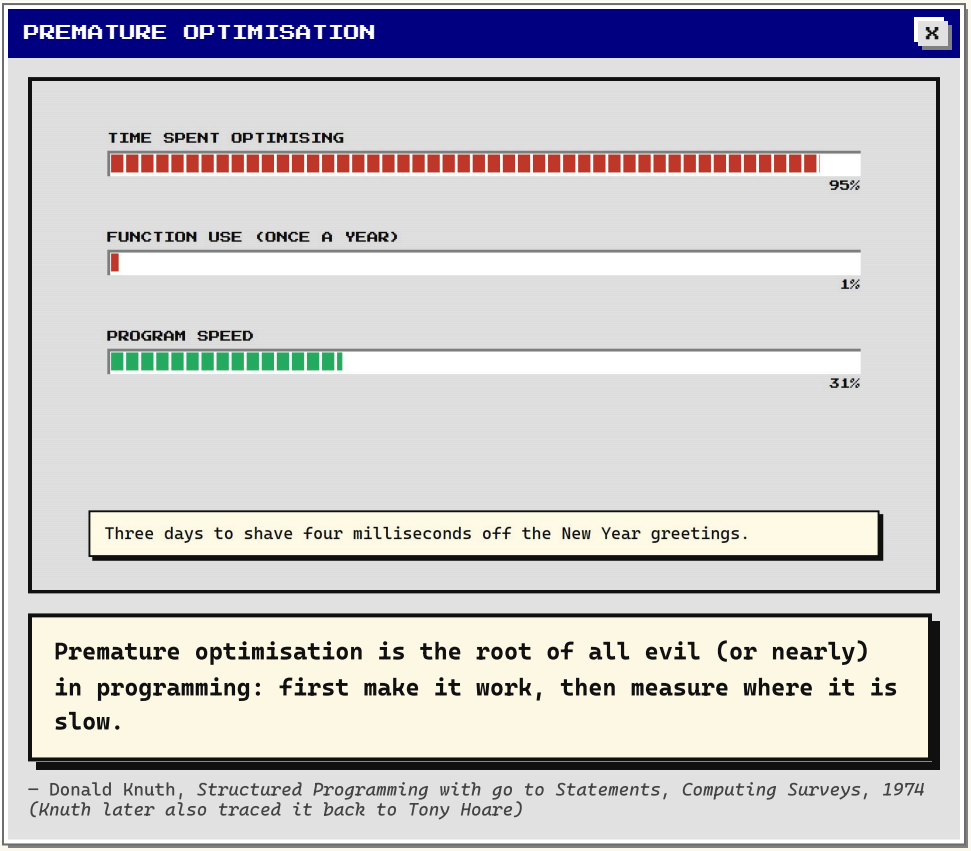
There are two ways of constructing a software design: make it so simple that there are obviously no deficiencies, or so complicated that there are no obvious deficiencies. The first method is far more difficult.

— C. A. R. Hoare, *The Emperor's Old Clothes, 1980 Turing Award lecture, published in Communications of the ACM, 1981*

Complexity is an excellent hiding place. In a fifty-line program a bug stands out like a fly on a white tablecloth. In a fifty-thousand-line one it has a house, a family and an allotment.

Hoare reminds us that simplicity isn't laziness: it's the hardest thing to achieve. You have to remove, rethink, throw away code you'd grown fond of. Adding is easy; giving things up takes courage.

I asked a colleague whether his module had any defects. He replied: 'None that you can see.' It was true. You couldn't see anything. Not even the logic. It was a masterpiece of Lombard fog in the form of code, and underneath, in the fog, the bugs were playing bowls.



Premature optimisation is the root of all evil (or nearly) in programming: first make it work, then measure where it is slow.

— Donald Knuth, *Structured Programming with go to Statements*, *Computing Surveys*, 1974 *(Knuth later also traced it back to Tony Hoare)*

There's a type of programmer who spends three days making a function lightning fast when it is called once a year, on 1 January, to print a New Year greeting. He gains four milliseconds. He celebrates as if it were the Moon landing.

The point of the phrase is common sense: you don't know where the slowness is until you measure. Optimising blindly makes the code unreadable and leaves the real bottleneck untouched, which is usually a query run inside a loop.

My uncle optimised his cellar: shelves in alphabetical order, indexes, a cache of his favourite bottles. Then he realised the door was locked and the key was in the cellar. Access time was infinite, but very well organised.

PIKE'S RULES

PROFILE.TXT

```
Guess: slow at computing
Measured:
  compute        2%
  file reading  97%   (same file x 1000)
  other          1%
```

Measure first, then optimise. Gut feeling about speed is a horoscope.

You can't tell in advance where a program will spend its time: measure first. Fancy algorithms are slow when n is small, and it is the data, more than the code, that dictates the right structure.

— Rob Pike, *Notes on Programming in C, 1989*

There are five rules, and they sound like the commandments of a terminal monk: measure, don't guess; don't use complicated algorithms if n is small, and n is usually small; simplicity beats elegance; data rules.

The message is that a programmer's intuition about performance is about as reliable as a horoscope. You think you know where the slowness is, you spend two days optimising the wrong thing, and the real problem was a loop reading the same file a thousand times.

I implemented a self-optimising balanced tree to manage the office phone book. It contains seven numbers. The old scrap of paper stuck to the fridge finds them faster. The scrap of paper also has an elegant magnetic mount.

THE LAW OF DEMETER

ORDER.JS

```
name = client.order.address
         .city.county.region.name
// six links: if one changes, it all falls

better: name = client.county()
```

A chain of gossip: every piece knows too much about the others.

An object should only talk to its immediate neighbours and not to strangers: no chains of calls running across half the program.

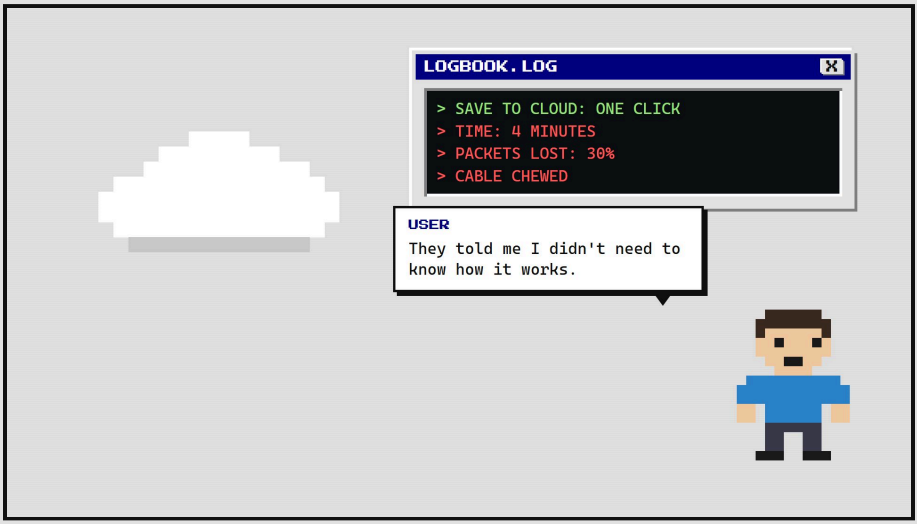
— Ian Holland, *Demeter Project, Northeastern University, 1987 (Karl Lieberherr's group)*

It's also known as the principle of least knowledge. In practice: if you want to pay the newsagent, you give him the money. You don't hand him your wallet so he can rummage through it, find the card, ring the bank and ask your mother for the balance.

In code, a line like customer.order.address.city.county.name is a chain of gossip: every link knows too much about the others, and if one changes they all collapse. Reducing what each piece knows makes them replaceable.

I applied the law at the office. I no longer ask a colleague to ask the technician to ask the supplier. I just ask the colleague. He doesn't know anything, but the system is much more decoupled. Now nothing works, but elegantly.

THE LAW OF LEAKY ABSTRACTIONS



All non-trivial abstractions, to some degree, leak the details they were supposed to hide.

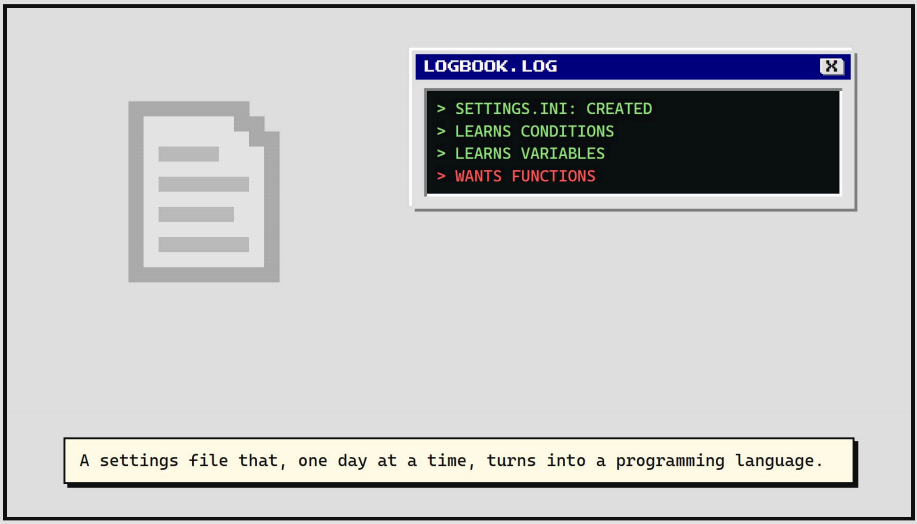
— Joel Spolsky, *The Law of Leaky Abstractions*, *Joel on Software*, 2002

An abstraction is a nice façade: 'Don't worry about how the network works, just save the file.' It works beautifully, until the day the file takes four minutes and you have to find out what latency is, what a lost packet is and what a cable chewed by a rodent looks like.

The point is that convenient tools save time in normal work, but they don't excuse you from understanding what's underneath. When they leak, and they always leak, you need someone who knows how to go down to the cellar.

The cloud is the supreme abstraction: 'Your data is in a cloud.' Then the cloud has a fault in a data centre with a name, an address and a broken air conditioner. The cloud has a postcode. I knew it. I've been there.

GREENSPUN'S TENTH RULE



Any sufficiently complicated C or Fortran program contains an ad hoc, slow, bug-ridden implementation of half of Common Lisp.

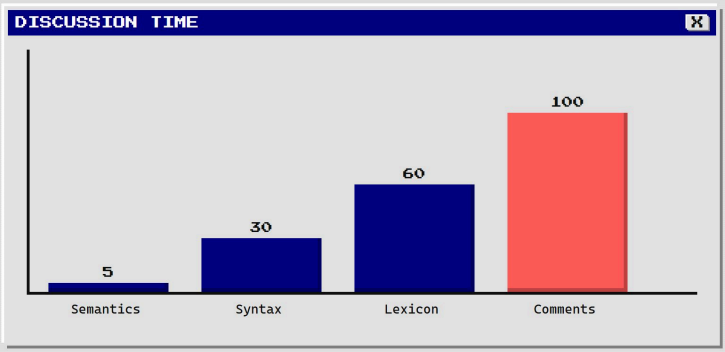
— Philip Greenspun, *Around 1993; the first nine rules never existed*

It's called the 'tenth' but it's the only one. The other nine were never written. That alone makes it the most honest rule in computing: it promises nine things and delivers none of them, like certain quotes.

The point is that large programs end up reinventing, badly, powerful tools that already existed: a mini configuration language, a home-made interpreter, a rule system written in a text file that at some point acquires loops.

My old management system had a settings file. Then it learnt conditions. Then variables. One day it asked to be allowed to declare functions. We switched it off in time. Now we keep it in a drawer, but at night I can hear it compiling.

WADLER'S LAW



The core of the language: five minutes. The comment symbol: three weeks.

In designing a language, the time spent discussing a feature grows as you move from semantics to syntax, then to lexical syntax, down to comments, which beat everything.

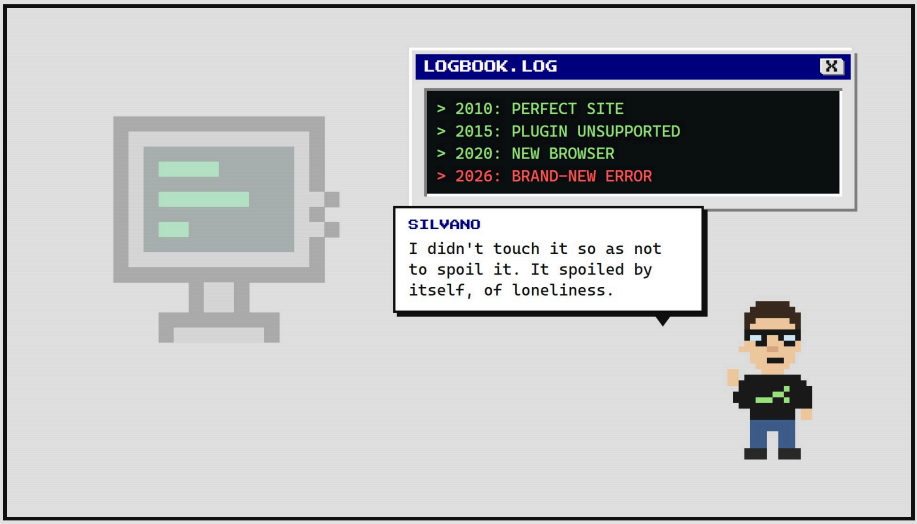
— Philip Wadler, *Messages on the Haskell mailing list, 1992 and 1996*

The deep meaning of a language? Five minutes, everyone agrees. The symbol for comments? Three weeks, two schisms and a fork. The smaller the question, the more everyone feels like an expert.

It's the programmers' version of a universal human rule: on difficult things people stay silent, on easy ones they go to war. Anyone who has taken part in a code review has seen fifty comments about spaces and none about the logic.

In our team we argued at length over tabs or spaces. We set up a committee. The committee produced a document. The document was indented with a mixture of both. Now the committee has a subcommittee. The project is progressing, in the comments.

LEHMAN'S LAWS



Software used in the real world must change continually or it becomes progressively less useful; and every change increases its complexity, unless someone works specifically to reduce it.

— Meir M. Lehman, *Laws of software evolution, formulated with László Bélády from the 1970s onwards*

Software is like a garden, not a statue. Leave it alone and it doesn't stay perfect: the weeds grow by themselves. The world around it changes, laws change, browsers change, and the program that was flawless in 2010 now asks you to install an extinct plugin.

The cruellest part is the second one: every change adds disorder. A patch here, a special case there, and the code becomes a block of flats with twelve different intercoms.

I have a Joomla site I haven't touched in years so as not to spoil it. I didn't spoil it. It spoilt itself, out of sheer loneliness. Now I visit it on Sundays, update an extension for it and it thanks me with a brand-new error.

THE SYSADMIN'S ADVICE

If you write code, even a little, even just for yourself, use a **version control system**. The most widespread is Git, and it's free. It works like a time machine with labels: every time the work is in a good state, you save a snapshot with a sentence explaining what you changed.

The advantages are huge. You go back to yesterday with one command. You see exactly which line changed, and when. You try a crazy idea on a separate branch and throw it away without damage. Goodbye to duplicated folders with the date in the name and the word ‘old’ repeated three times.

Three golden rules: save often and in small steps; write messages a stranger would understand, that is, you in six months; never put passwords or access keys in the repository, not even ‘for a moment’. That moment, on the Internet, lasts forever.

CHAPTER 7

CH 07 - MEETINGS

Projects, meetings and clients

Sooner or later every IT person discovers an awkward truth: the hardest problems are not in the computer. They are in the meeting room, in the quote, in the email with twelve people in copy. Code can be fixed. An estimate given in the corridor can't.

This chapter gathers the laws of teamwork. It starts with deadlines, always wrong from an excess of optimism, and moves on to clients, who know what they wanted only after seeing it. Then there are meetings, which multiply by recursion and always discuss the least important thing, and demos, which fail in proportion to the rank of whoever is watching.

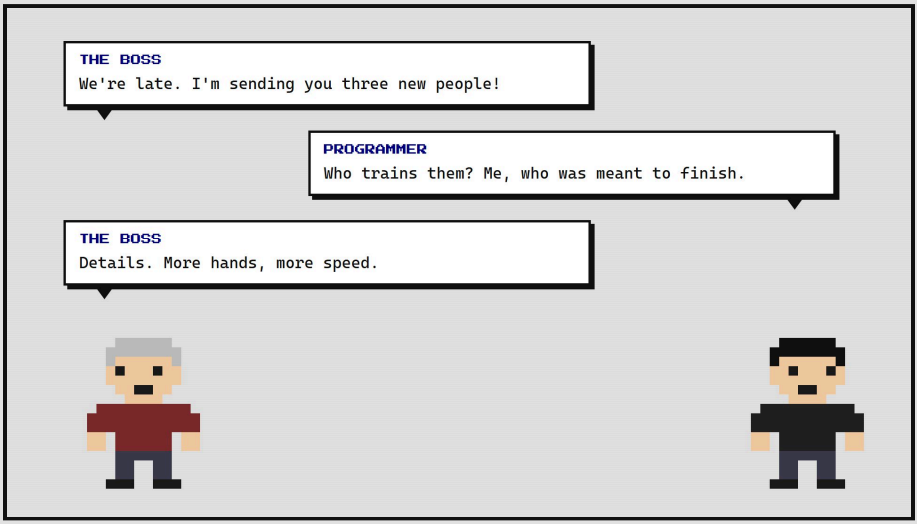
At the end come the bitterest laws, the ones about organisations: numbers that become targets and stop telling the truth, temporary fixes that hold the company up, systems that end up resembling the org chart, closed corridors included.

Anyone who works in an office will recognise colleagues, bosses and themselves. That's normal. The trick is to recognise yourself before the meeting, not during it.

CONTENTS.DIR

01 Brooks's Law
02 Law of the pi multiplier
03 Law of the sworn estimate
04 Law of the clairvoyant client
05 Law of the phantom requirement
06 Law of the recursive meeting
07 Parkinson's law of triviality
08 Sayre's Law
09 Wiio's laws
10 Law of the demo in front of the client
11 Corollary of the important audience
12 Law of the eternal temporary fix
13 Conway's Law
14 Gall's Law
15 Weinberg's Law
16 Goodhart's Law
17 Campbell's Law
18 The Dilbert principle

BROOKS'S LAW



Adding people to a late software project makes it later.

— Fred Brooks, *The Mythical Man-Month*, 1975

The visionary manager looks at the Gantt chart, sees red and proclaims: 'Let's double the team!' It's the same reasoning by which nine women could have a baby in one month.

The problem is that newcomers have to be trained, and the people training them are exactly the ones who were finishing the work. Then every extra person multiplies the conversations: four people talk to each other, twelve hold meetings to decide when to hold meetings.

Once, to make up for a delay, I was given three interns. On day one I explained where the toilets were. On day two, the Wi-Fi password. By day three the project was another two weeks late, but the Wi-Fi was working beautifully.

LAW OF THE PI MULTIPLIER



To get the real duration of a software project, take the programmer's estimate and multiply it by pi. If the project is important, use the decimal places too.

— Anonymous, *computing folklore*

A programmer estimating a job is an optimist by constitution. They imagine a world without surprises, without meetings, without the client ringing to ask whether the logo could be a bit bigger. In that ideal world, 'two days' means two days.

Then reality turns up, and reality has its own maths. Two days become six point two eight, and the nought point two eight is the moment you discover the test server's password belongs to someone who retired.

Why pi, of all things? Nobody knows. But it is an irrational and infinite number, so it is the best one for describing deadlines. I've stopped writing days in my quotes. I just write the diameter of the project and let the client do the sums.

LAW OF THE SWORN ESTIMATE



Any estimate given verbally, with a thousand caveats and three 'maybes', will become a deadline written in the boss's calendar.

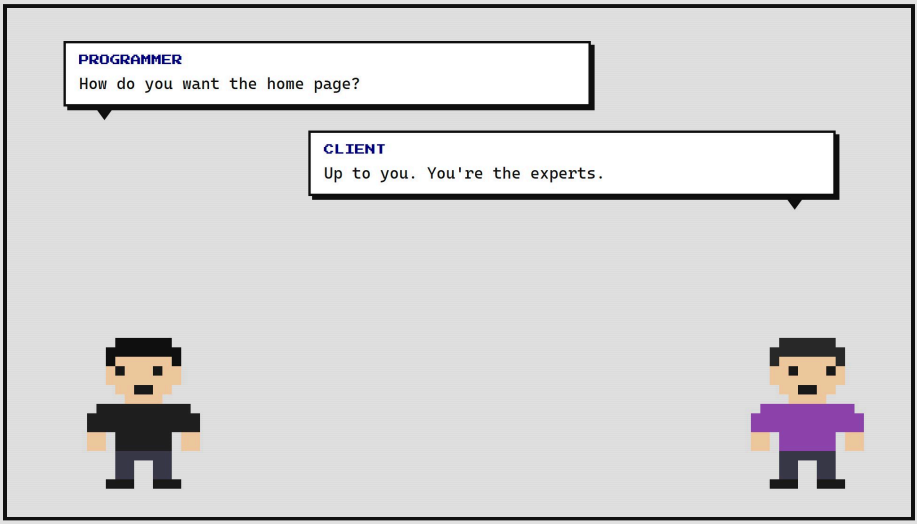
— Anonymous, *computing folklore*

Questioned in the corridor, the programmer answers with diplomatic caution: 'Roughly, if all goes well, maybe, with no surprises, a week. But don't hold me to it.' The boss nods. He only heard 'a week'.

The next day that sentence appears in an official document, then in an email to the client, then in a colourful chart projected in a meeting. The caveats were removed during compression, like the details of a photo sent by phone.

The visionary manager, meanwhile, presents it with pride: 'The team has committed.' The team didn't know it had committed. Since then, when someone asks how long something will take, I answer with a question: 'Is this a question or a contract?' The silence that follows usually counts as an answer. And as a quote.

LAW OF THE CLAIRVOYANT CLIENT



The client knows exactly what they want at the precise moment you deliver what they asked for.

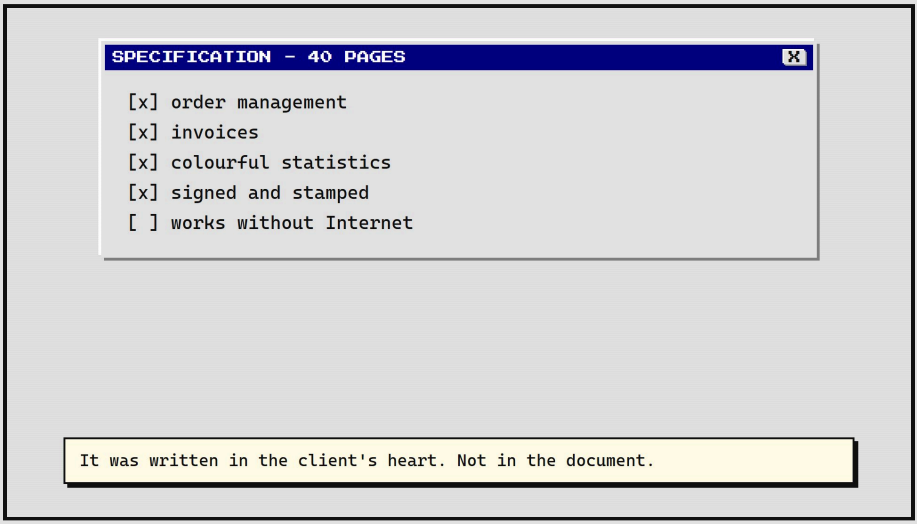
— Anonymous, *computing folklore*

During analysis the client is vague, pensive, zen. 'You decide, you're the experts.' Then they see the finished work and have a mystical revelation, as if the heavens had opened: now they know. They know it wasn't this.

The phenomenon is well known in technical literature too: there's a famous anonymous cartoon showing a tree swing as the client wanted it, as the salesman understood it, as the programmer built it. None of the versions is a swing.

I've found a foolproof method. I knock up a wrong version straight away, on purpose, in half a day. The client looks at it, is horrified and finally speaks clearly. It is the only case in computing where the bug is the most useful feature of the project. I invoice it as 'requirements analysis by contrast'.

LAW OF THE PHANTOM REQUIREMENT



The most important requirement of the project is communicated verbally, in passing, on delivery day.

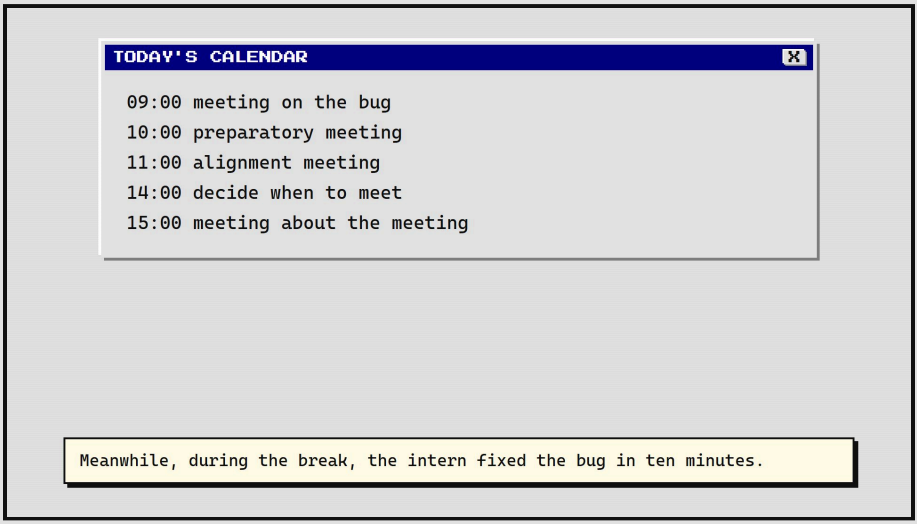
— Anonymous, *computing folklore*

The specification is forty pages long, with tables, charts and a table of contents that reads like a peace treaty. Everything has been discussed, signed, stamped. Then, on delivery day, while everyone is shaking hands, the client adds: 'Oh, of course it has to work without the Internet too.'

They don't say it out of malice. They say it the way you say 'good morning', with the serenity of someone who thinks they're mentioning an obvious detail. For them it was always like that. It was written in their heart, if not in the document.

Since then I've introduced the meeting of the final question. Before signing, I ask: 'Is there anything you think is so obvious you haven't mentioned it?' The silence that follows is always long. And inside that silence, every time, lives a new module to develop.

LAW OF THE RECURSIVE MEETING



Every complex technical problem generates a meeting, which generates a preparatory meeting, which generates the need for a meeting to decide when to hold the meeting.

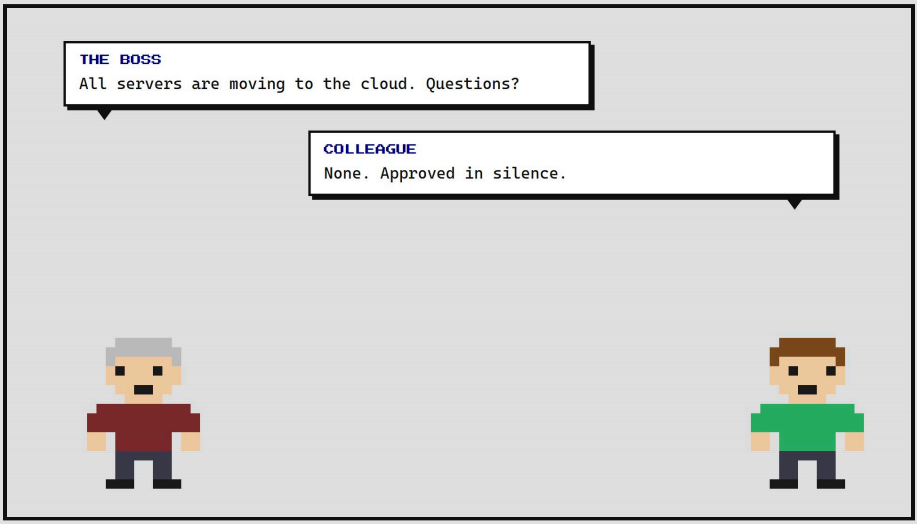
— Anonymous, *computing folklore*

In computing, recursion is a function that calls itself. In companies it is a meeting that convenes itself, forever, until someone goes on holiday and breaks the loop through calendar exhaustion.

The pattern never changes. Eight people meet to discuss a bug. It turns out more information is needed. A new meeting is set with four other people. At that meeting the need to 'align' emerges. Meanwhile the intern has fixed the bug, in ten minutes, during the break.

There's always the manager who opens proceedings solemnly: 'Gentlemen, today we decide.' And indeed something is decided: the date of the next meeting. I suggested installing a counter in the meeting room, like the ones at village fêtes. Every hour of meeting, one line of code fewer.

PARKINSON'S LAW OF TRIVIALITY



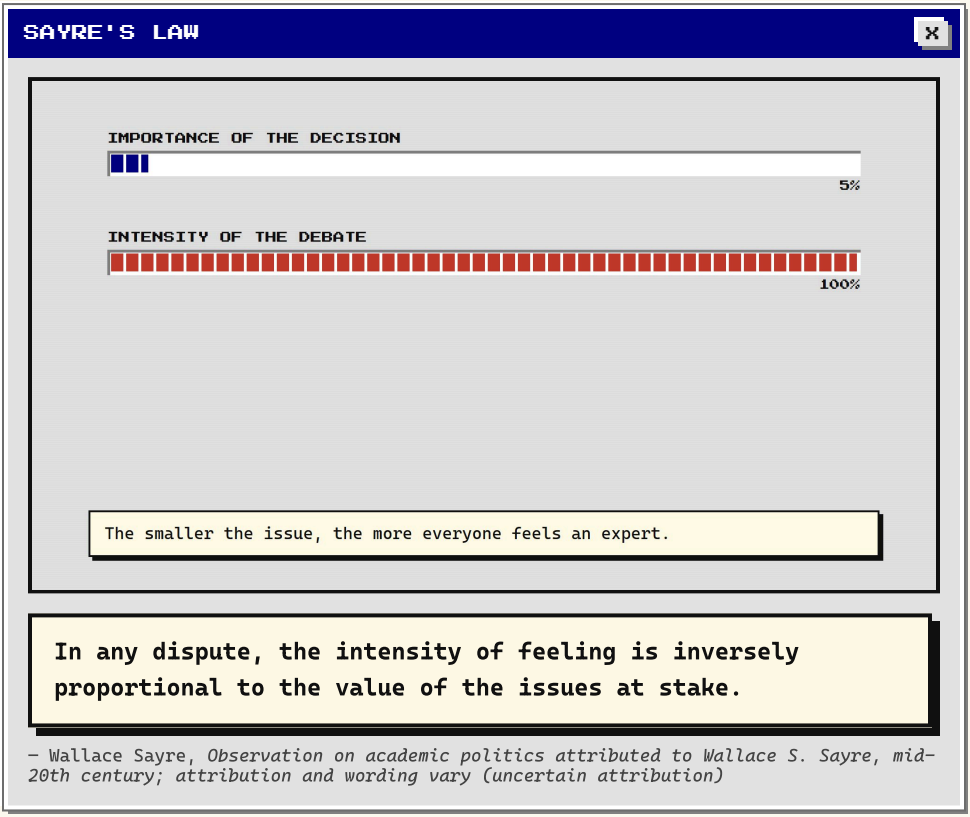
An organisation gives time to decisions in inverse proportion to their importance: little is said about the reactor and a great deal about the bike shed.

— C. Northcote Parkinson, *Parkinson's Law, 1957; popularised in computing as 'bikeshedding' by Poul-Henning Kamp in the FreeBSD community, 1999*

The board approves the multimillion project, which nobody understands, in two minutes. Then it spends an hour on the bike shed, which everyone understands perfectly and on whose colour everyone has an opinion.

In computing it has become a watchword: when a discussion flares up over an irrelevant detail, someone writes 'bikeshedding' and everyone, for a moment, feels ashamed. Then they go back to arguing about the variable name.

At my last meeting we silently approved moving all the servers to the cloud. Then we argued for forty minutes about the icon for the new management system: a cog or a cloud? In the end we chose a cloud-shaped cog. Nobody knows what it does. Much like the cloud, really.



In any dispute, the intensity of feeling is inversely proportional to the value of the issues at stake.

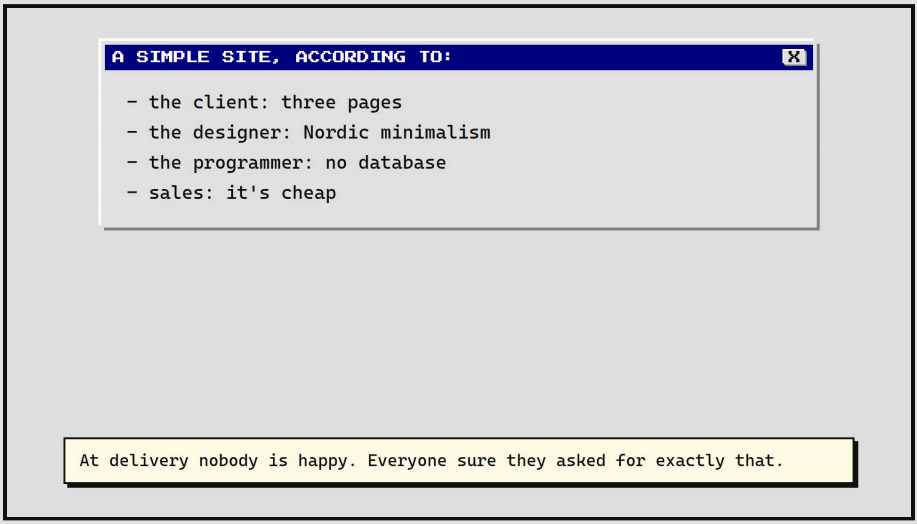
— Wallace Sayre, *Observation on academic politics attributed to Wallace S. Sayre, mid-20th century; attribution and wording vary (uncertain attribution)*

Born to describe squabbles between university professors, it applies perfectly to technical meetings. Migrating the entire database goes through in five minutes. The colour of the 'Cancel' button triggers a two-hour debate and an email with half the company in copy.

The reason is simple: on huge questions few people feel competent; on tiny ones everyone has an opinion, and defending it costs nothing.

The longest meeting of my career was about the name of a shared folder. 'Documents', 'Documentation' or 'Docs'. Four hours. We voted. 'Documents_FINAL' won. The next day 'Documents_FINAL_new' appeared. The conflict became permanent, like certain cold wars, but with more write permissions.

WIIO'S LAWS



Communication usually fails, except by accident. And if it seems to have succeeded as intended, there is certainly a misunderstanding.

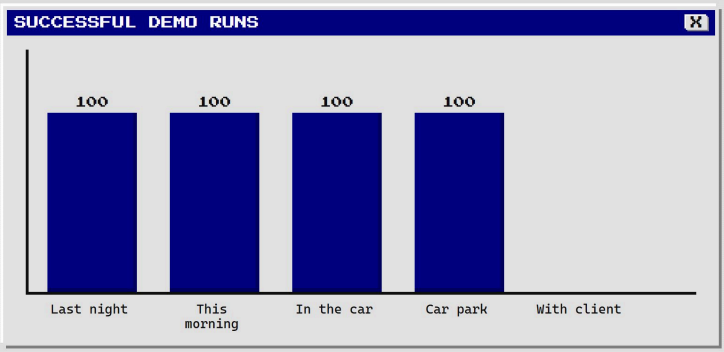
— Osmo Antero Wiio, *'Wiion lait – ja vähän muidenkin'*, 1978

A Finnish professor of communication reached a conclusion every sysadmin already knows: communication doesn't work. When it does, it's an accident. And if it seems everyone has understood, it's only because each of them has understood something different.

The client asks for 'a simple website'. He means three pages. The designer means Nordic minimalism. The programmer means no database. The salesman means 'cheap'. On delivery nobody is satisfied, but everyone is sure they asked for exactly that.

That's why written specifications, minutes and recap emails exist. They serve to document the misunderstanding precisely. Now I end every meeting by asking everyone to repeat what we've decided. Every time, five projects come out. At least we find out straight away, and not at acceptance testing.

LAW OF THE DEMO IN FRONT OF THE CLIENT



Success rate. The last reading was taken in the presence of a decision-maker.

A program that has worked in a hundred trials fails on the hundred and first: the one in front of the client.

– Anonymous, *computing folklore (also known as the 'demo effect')*

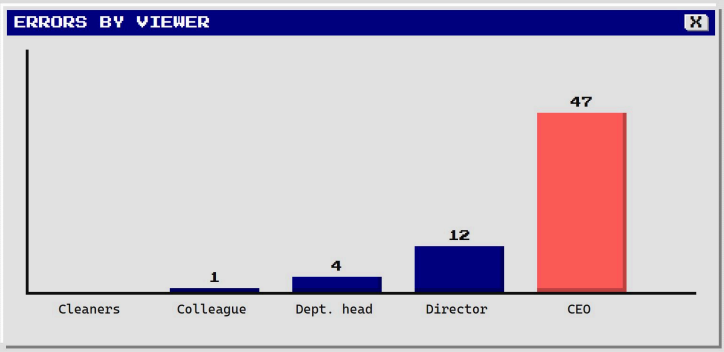
The demo was rehearsed the night before, in the morning, in the car and in the car park. All smooth. Then you walk in, the client sits down, you say confidently 'Now let me show you' and the screen becomes a palette of errors never seen before.

IT people call it the demo effect. It is the only known phenomenon capable of detecting the presence of a decision-maker ten metres away, through walls.

The best part is that as soon as the client leaves, the program runs perfectly again. It looks at you as if to say: 'What happened? I was there.'

Advice from someone who's lived it: always prepare a plan B. A recorded video of the working demo, or screenshots of the key steps. If the program throws a tantrum, you show the film.

COROLLARY OF THE IMPORTANT AUDIENCE



A hierarchy, as precise as an org chart. With the cleaners, the server calms down.

The higher the rank of the person watching the demonstration, the more likely the demonstration is to fail.

— Anonymous, *computing folklore (Usenet collections of computer laws)*

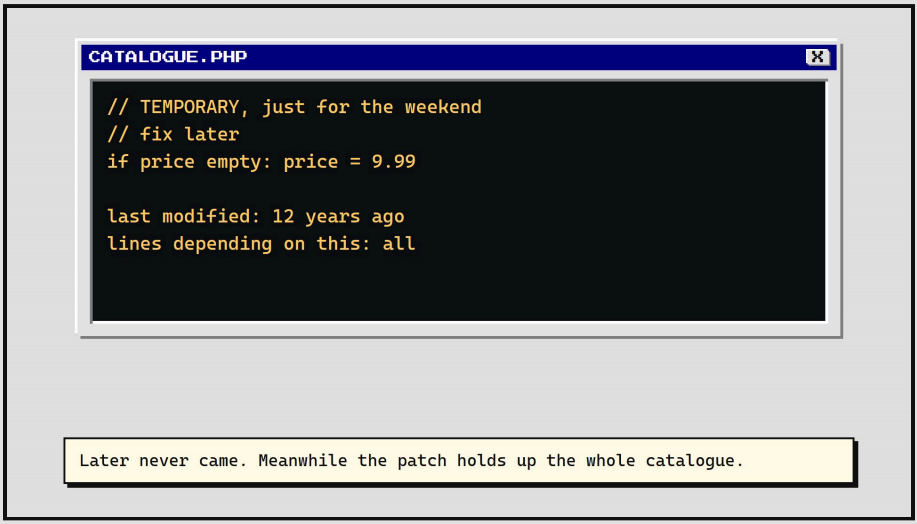
In front of a colleague, the demo holds up. In front of the head of department, it hesitates. In front of the director, it coughs. In front of the CEO who has just arrived by helicopter, the computer switches off and asks to install forty-seven updates.

It's a hierarchy of failure, as precise as an org chart. Some researchers maintain that expensive suits emit an electromagnetic field. I have no proof. But I've noticed the server calms down when the cleaner comes in.

The visionary manager, meanwhile, smiles and says: ‘Excellent, we've identified an opportunity for improvement.’ Which in corporate language means: see you in my office.

Real advice: for important presentations use a stable, ready-prepared environment, and do a full rehearsal in the same room, on the same computer.

LAW OF THE ETERNAL TEMPORARY FIX



In software, nothing is more permanent than a temporary solution.

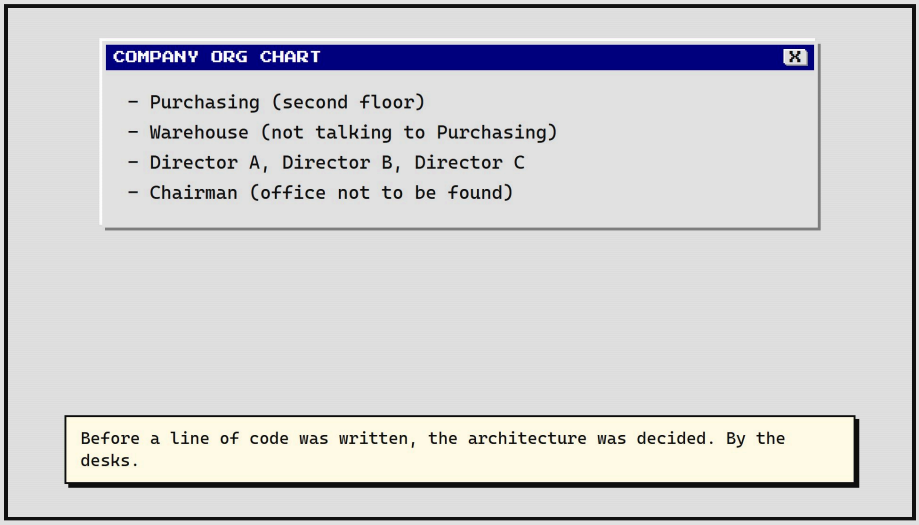
— Anonymous, *computing folklore*

Every program contains a line with the note 'tidy this up later'. Later never comes. The line survives three versions, two server moves and a change of company ownership.

Temporary solutions have an extraordinary constitution. Permanent solutions, carefully designed, get rewritten every two years. The patch applied at eleven at night to get the invoices going again, on the other hand, becomes heritage, with its own dignity and its own aura.

I look after a website where, twelve years ago, someone wrote a little script 'just for the weekend'. Today it holds up the entire catalogue. Nobody dares touch it. Now and then I look at it, quietly, with respect, the way you look at the load-bearing wall of an old farmhouse: crooked, yes, but take it out and the whole thing comes down.

CONWAY'S LAW



An organisation that designs a system will end up producing a system that copies its own communication structure.

— Melvin Conway, *How Do Committees Invent?*, *Datamation*, *1968*

If four groups work on a compiler, you get a four-pass compiler. If purchasing doesn't talk to the warehouse, the management system will have two product registries that don't say hello to each other.

Software is a mirror. You look at the architecture and see the org chart, grudges and all. That module that doesn't communicate with anyone? It's the colleague on the third floor who eats lunch alone.

I once saw a website with three different menus, one for each director. A fourth menu, hidden, led to the chairman's page. Nobody knew how to get there, exactly like his office. The architecture was perfect: it mirrored the company down to the last locked corridor.

GALL'S LAW



A complex system that works has always evolved from a simple system that worked. A complex system designed from scratch never works and cannot be patched up to make it work.

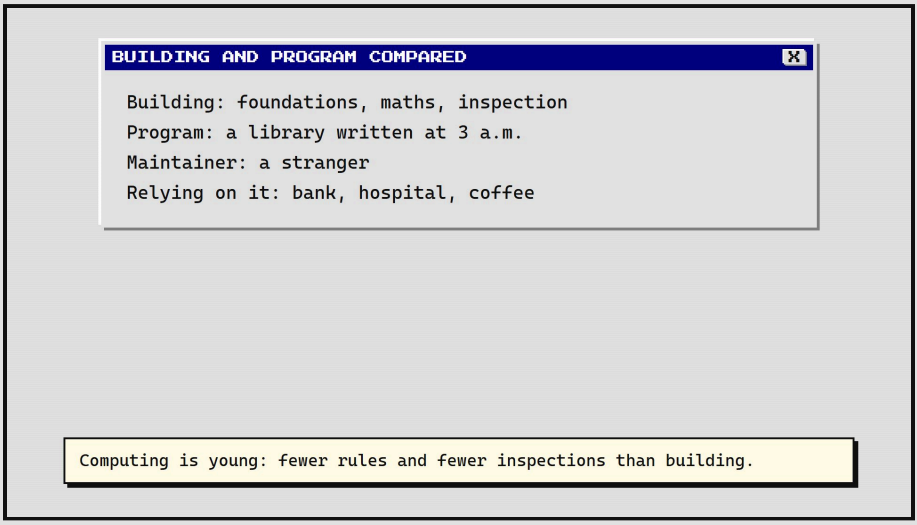
— John Gall, *Systemantics (General Systemantics)*, 1975

The digital transformation guru takes the stage: 'We'll rebuild everything from scratch, microservices, artificial intelligence, blockchain, in six months!' Applause. Sixteen months later everyone goes back to the shared Excel spreadsheet that had been working since 2004.

The law is a hymn to evolution. The systems that hold up grew a little at a time, like stalactites or folders called 'New folder (7)'. The ones born huge and perfect on paper collapse, because nobody ever saw them working small.

My favourite management system started life as a text file with three lines. Today it runs an entire company. It still has those three lines inside, somewhere. Nobody touches them. They are the load-bearing relic.

WEINBERG'S LAW



If builders built buildings the way programmers write programs, the first woodpecker that came along would destroy civilisation.

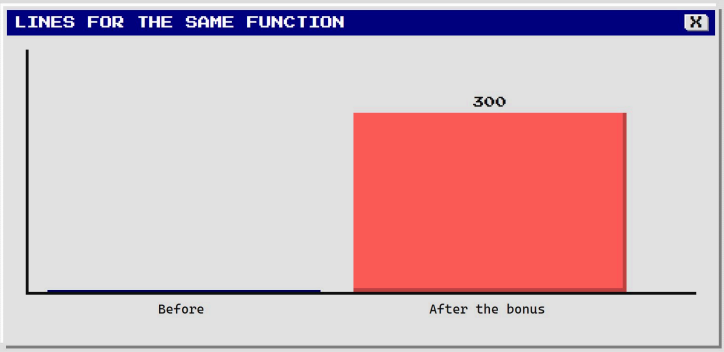
— Gerald Weinberg, *Uncertain attribution: appears in a 1975 collection (CoEvolution Quarterly) and as 'Weinberg's Second Law' in Arthur Bloch's Murphy's Law, 1977*

A real building has foundations, structural calculations, inspections. The average program has a library downloaded at three in the morning, maintained by a stranger, on which a bank, a hospital and the coffee machine all depend.

The law isn't just a dig: it reminds us that computing is a young discipline, with fewer regulations and fewer inspections than construction. Much of what we use holds up through goodwill and luck.

Once I saw a woodpecker on the roof of our warehouse, right above the server room. I stood between it and the aerial. We looked at each other for a long time. In the end it left. I think it understood it wasn't necessary: the server would break by itself, on Monday.

GOODHART'S LAW



Lines of code get rewarded. Productivity triples. The software, mysteriously, doesn't.

When a measure becomes a target, it ceases to be a good measure.

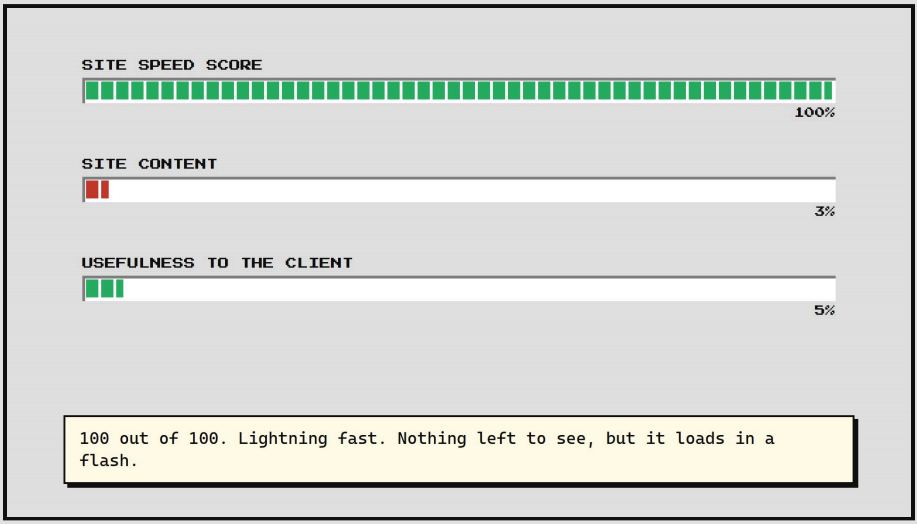
— Charles Goodhart, *Paper on monetary policy, 1975; the famous wording is Marilyn Strathern's, 1997*

The company decides to measure programmer productivity by counting lines of code. The following month the programs are enormous. A function that used to fit in three lines now takes three hundred, with poetic comments and four-word variable names. Productivity has tripled. The software, mysteriously, hasn't.

That's Goodhart's Law: the moment you reward a number, people learn to inflate the number instead of improving the thing it was supposed to represent. It applies to closed tickets, to review stars, to website speed scores.

In the support department they measured how quickly calls were closed. One colleague closed them all in eight seconds. He answered, said 'try restarting it' and hung up. Employee of the month three times. Users called him the Lightning Bolt.

CAMPBELL'S LAW



The more a quantitative indicator is used for important decisions, the more it is corrupted, and the more it distorts the processes it was meant to monitor.

— Donald T. Campbell, *'Assessing the Impact of Planned Social Change', 1976 (earlier formulations from 1969)*

A close cousin of Goodhart's Law, Campbell's Law adds a gloomy detail: the number not only stops telling the truth, it starts warping the work itself. If the bonus depends on the score, the job becomes producing the score.

In the world of websites it's glaringly obvious. It's decided that only the speed tool's rating matters. So out go the images, the text, the features, everything that weighs anything. The site scores a hundred out of a hundred. It's lightning fast. There's nothing left to see, but it loads in a flash.

A company I know assessed its IT person by the number of computers working at the end of the month. He, very wisely, touched none of them on the last day of the month. They all worked. Painfully slow, full of warnings, but statistically in perfect health.

THE DILBERT PRINCIPLE



LOGBOOK.LOG

```
> SERVER ACCESS: DENIED
> NEW OFFICE: WITH A VIEW
> ROLE: MANAGER
```

Primitive access control: instead of permissions, an office with a view.

Companies tend to promote their least competent employees into management, to limit the damage they can do to the real work.

— Scott Adams, *Wall Street Journal*, 1995; book *'The Dilbert Principle'*, 1996

The Peter Principle said you rise until you become incompetent. The Dilbert principle is more cynical: some people start out incompetent and are moved upstairs on purpose, where they can't touch anything important. Like making the hyperactive child team captain, so he doesn't touch the ball.

In computing the phenomenon has an unofficial technical name: the visionary manager. He doesn't know what a database is, but he's understood that the future lies in data. He says so in meetings, banging his fist on the table. Then he asks someone to print out the email for him.

Let's admit it, the system has its own efficiency: the person is kept away from the server, and the server is grateful. It's a primitive form of access control. Except that instead of permissions it uses an office with a view.

THE SYSADMIN'S ADVICE

Two habits that have saved more projects for me than any software.

The summary email. After every important meeting or call, write a short email to everyone present within an hour: what we decided, who does what, by when, what is still open. Five lines. If someone understood something else, you find out at once and not at delivery. And in three months, when nobody remembers anything, that email will be the official version of events.

The range estimate. Never give a single number. Say ‘three to six days, because it depends on this and that’. A single number becomes a promise; a range forces everyone to talk about what makes it widen. And put in writing what is excluded from the job, too. The phantom requirement hates bullet points.

NOTICE

This page has been intentionally left blank.

This is a Murphy's law too: the blank page always turns up where it isn't needed.

CHAPTER 8

CH 08 - FUTURE

Progress, updates and artificial intelligence

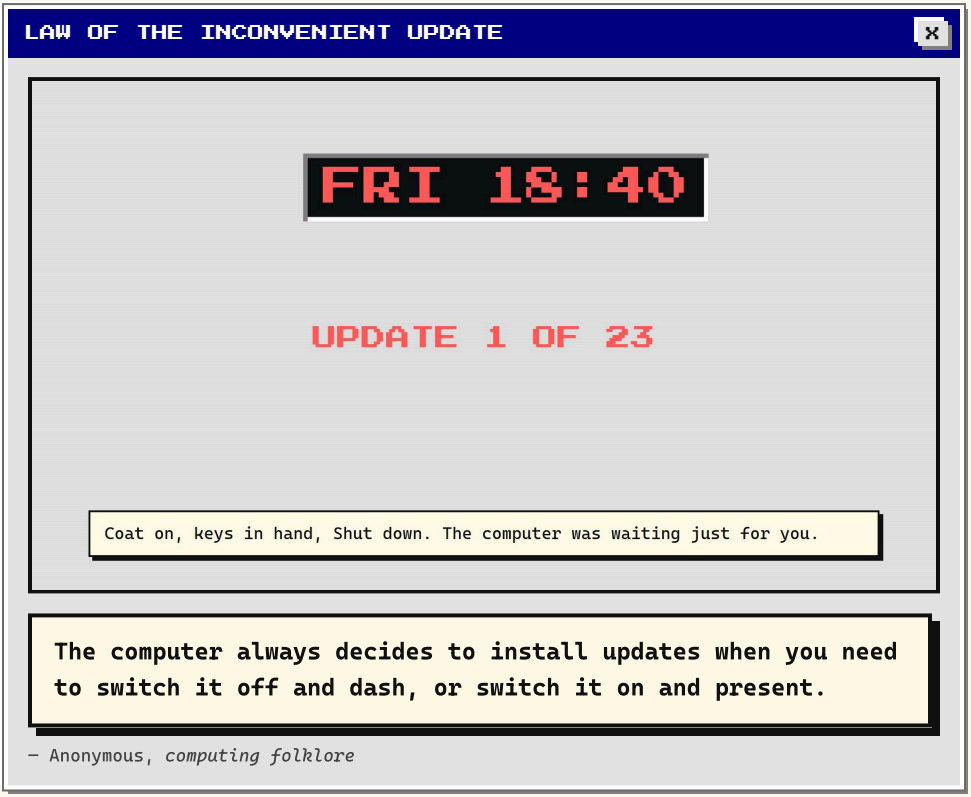
The last chapter looks ahead, which in computing means staring at an update's progress bar. That is exactly where we start: updates that arrive at the wrong moment, that fix one thing and break another, new computers already old as they leave the shop.

Then we scale up. There are the laws describing technology's great race: ever more powerful chips, ever more expensive factories, ever heavier software, programs that want to do everything. They are economic and engineering laws, sometimes optimistic, often amused. They explain why the computer is enormously faster than twenty years ago and the word processor opens in the same time.

The chapter closes with six corollaries devoted to artificial intelligence. These I wrote myself, after many hours working with these assistants, and I sign them with the pride of someone who has seen a machine apologise several times for the very same mistake. I am not against AI: I use it every day. That is exactly why I love it the way you love a brilliant colleague who now and then invents function names.

CONTENTS.DIR

01 Law of the inconvenient update
02 Law of the update that breaks things
03 Law of the new computer
04 Wirth's Law
05 Zawinski's Law
06 Atwood's Law
07 Rock's Law
08 Grosch's Law
09 Amdahl's Law
10 Gustafson's Law
11 Koomey's Law
12 Bell's Law of computer classes
13 The Jevons paradox
14 Amara's Law
15 Corollary of the imaginary function
16 Corollary of artificial apologies
17 Corollary of the three seconds
18 Corollary of the solemn prompt
19 Corollary of the rest unchanged
20 Corollary of the forgotten rule



The computer always decides to install updates when you need to switch it off and dash, or switch it on and present.

— Anonymous, *computing folklore*

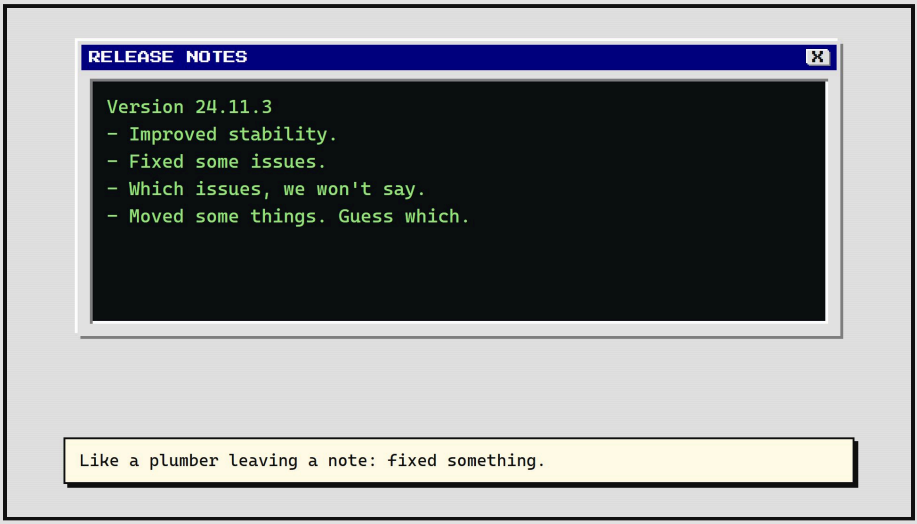
The computer spent the night switched on, idle, bored. It could have updated in peace. But no: it waits until you press ‘Shut down’ with your coat on and your keys in your hand. Then it announces, politely: ‘Update 1 of 23. Do not turn off your computer.’

And there you stand, as if at a level crossing. The update train goes slowly by, carriage after carriage. Sometimes it stops at thirty per cent to reflect on the meaning of life.

A neighbour of mine bought a new laptop to get his tax return done quickly. He switched it on in March. In May he filled in the return by hand, on paper, with his grandfather's fountain pen.

A technician's advice: set the system's active hours so updates install when you're not using the computer. And never force it off during an update.

LAW OF THE UPDATE THAT BREAKS THINGS



Every update that fixes one problem introduces at least one new one, usually in something that used to work perfectly.

— Anonymous, *computing folklore*

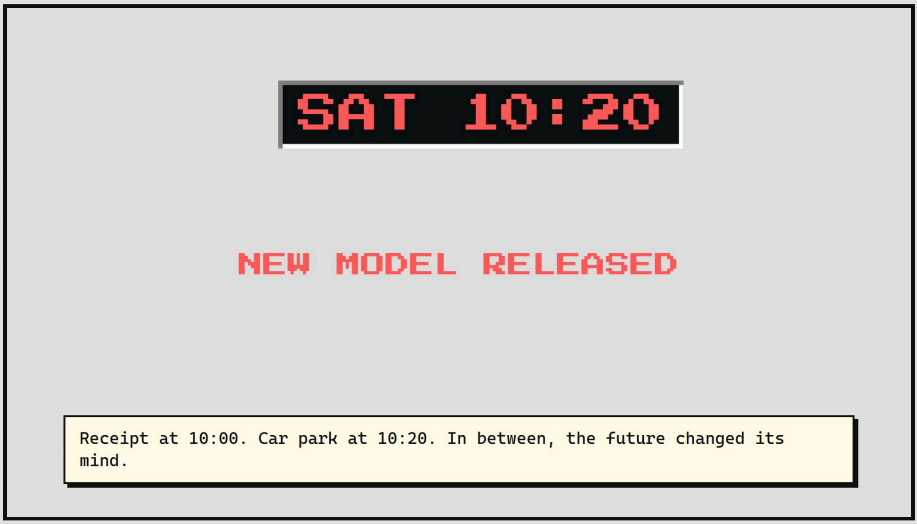
The update arrives with a list of exciting new features: ‘Improved stability. Fixed some issues.’ Which issues, it never says. It's rather like a plumber leaving a note: ‘Fixed something.’

The next day the printer won't print, the accounts program wants a licence and the recycle bin icon has turned green. They call it progress. I call it moving house: everything's still there, but nothing is where it should be.

The high priest of ‘we've always done it this way’ seizes the moment and proclaims: ‘I told you so, never update anything.’ Then he catches a virus through a program stuck in 2012.

Real advice: security updates must be installed. But before big updates, make a backup and, if you can, wait a few days for others to find the problems.

LAW OF THE NEW COMPUTER



The computer you've just bought has already been overtaken by the model released while you were driving home, which is cheaper too.

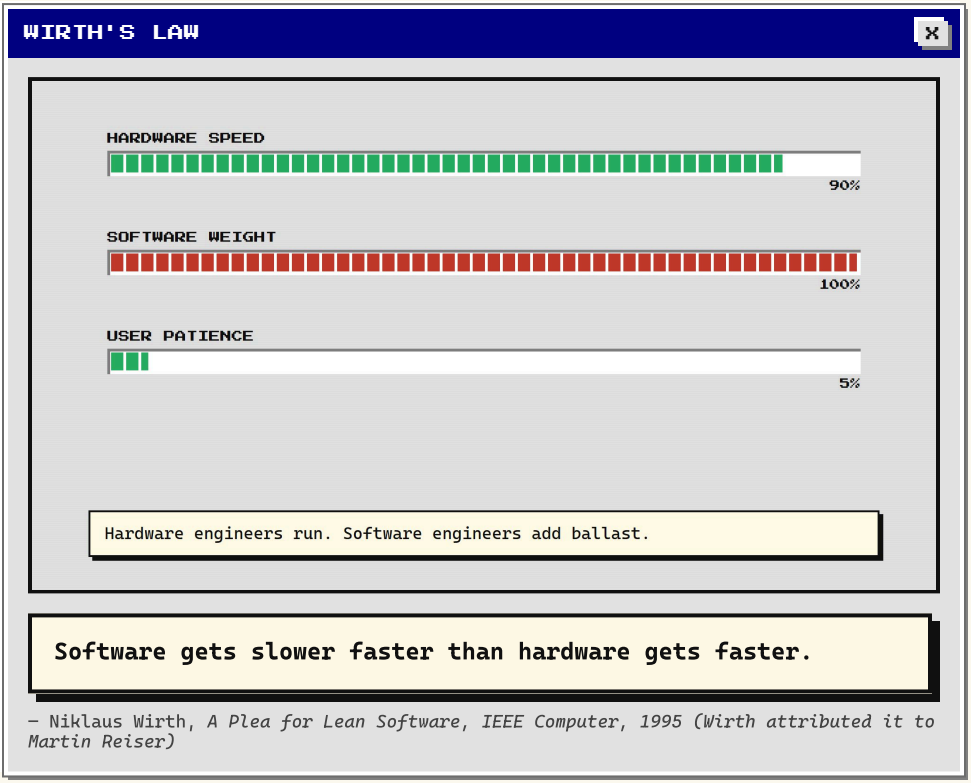
— Anonymous, *computing folklore*

The happiest moment in a computer's life is the receipt. The second happiest never comes, because on the way from the shop to the car the new model comes out: faster, thinner and on launch offer.

An acquaintance of mine, to beat the law, always waits for the next model. He has been waiting since 2011. He still uses the laptop from back then, which takes twelve minutes to boot. He says he's in no hurry: he's ‘investing in the future’.

Brothers and sisters of progress, here is the truth: no computer is ever up to date. There are only computers that don't yet know they're old.

Real advice: buy when you need to, sizing for three to five years of use. The perfect model will always come out tomorrow.



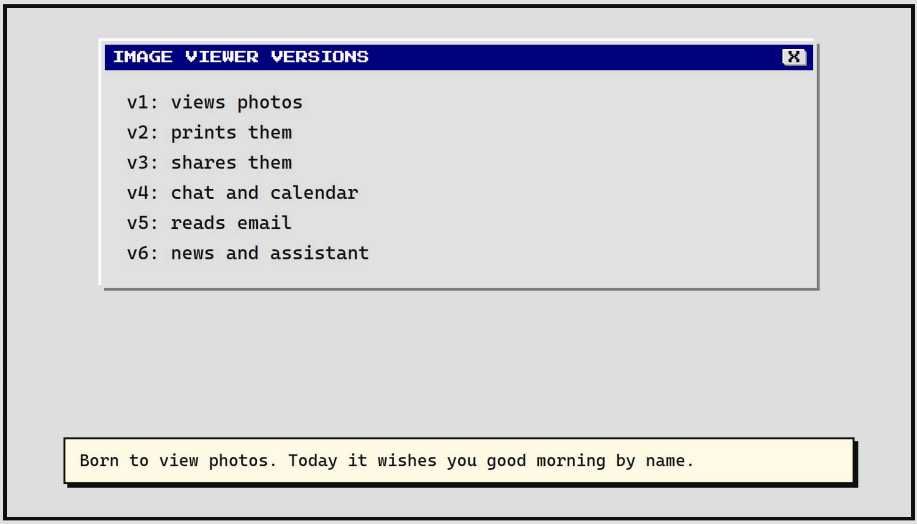
— Niklaus Wirth, *A Plea for Lean Software*, *IEEE Computer*, 1995 (Wirth attributed it to Martin Reiser)

In 1995 a word processor fitted on a few floppies and started in a flash. Today we have processors thousands of times faster, and the text editor takes just as long, because in the meantime it has learnt to load an entire browser, three analytics engines and an assistant asking if you'd like some help.

It's a race: hardware engineers run, software engineers add ballast. The finish line always moves a few seconds of loading further away.

My new PC has 32 gigs of RAM. I fed them to the calculator. Now it's lightning fast at working out 2+2, after updating itself, checking its licence and offering me a premium subscription for decimal numbers.

ZAWINSKI'S LAW



Every program attempts to expand until it can read mail. Those programs which cannot so expand are replaced by ones which can.

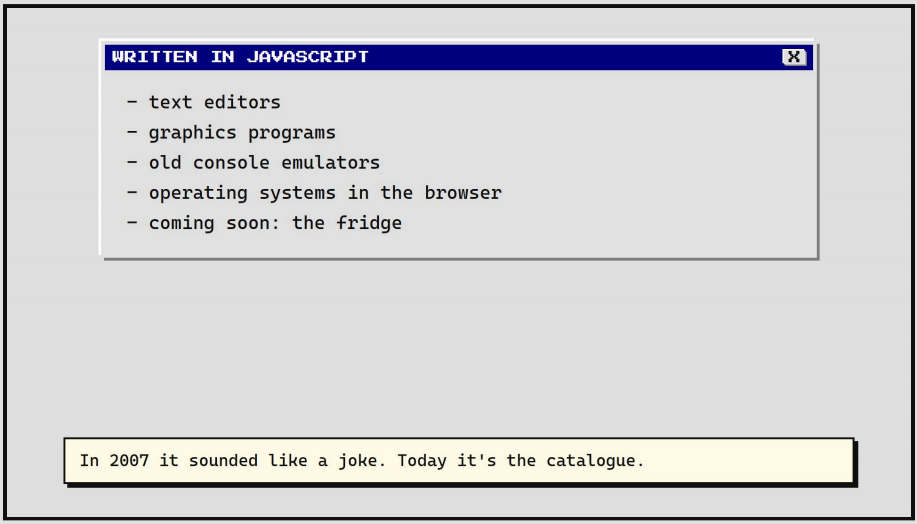
— Jamie Zawinski, *Observation attributed to Jamie Zawinski, Netscape developer, mid-1990s*

It starts life as an image viewer. Then it adds printing, then sharing, then chat, then a calendar. Two years later it has email, a news feed and a virtual assistant that greets you by your first name.

The meaning is serious: programs tend to pile up features because every user asks for one more, and anyone who stops looks outdated. Today email has been replaced by chat, video and artificial intelligence, but the fever is the same.

Yesterday I updated my calculator. Now it has an inbox. I've received a newsletter from the calculator informing me about the calculator's new features. I still haven't worked out where you do the sums.

ATWOOD'S LAW



Any application that can be written in JavaScript will eventually be written in JavaScript.

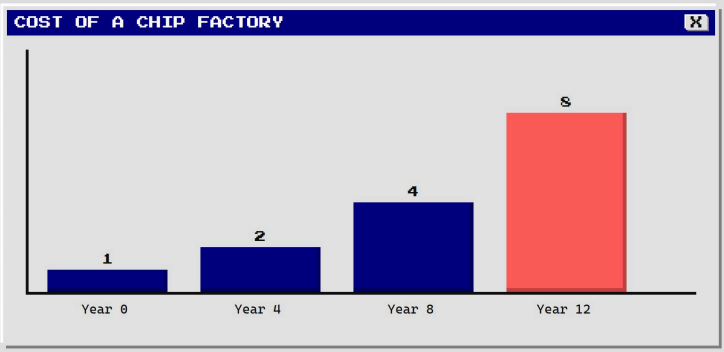
— Jeff Atwood, *The Principle of Least Power*, *Coding Horror blog*, 2007

In 2007 it sounded like a joke. Today JavaScript gives us text editors, graphics programs, emulators of old consoles, entire operating systems running inside a browser tab. The joke has come true like a local prophecy.

The point is that the most widespread and easily distributed technology tends to eat everything, even when it isn't the best fit. If it runs everywhere without installing anything, sooner or later someone will put the fridge in it too.

I've seen a programmer write a JavaScript simulator of an 1980s pocket calculator, complete with a battery that runs down. It weighs forty megabytes. The real calculator weighed eighty grams. Progress is measurable, mainly in megabytes.

ROCK'S LAW



Double every four years. The zeros now look like an IPv6 address.

The cost of a semiconductor fabrication plant doubles roughly every four years. Also known as Moore's second law.

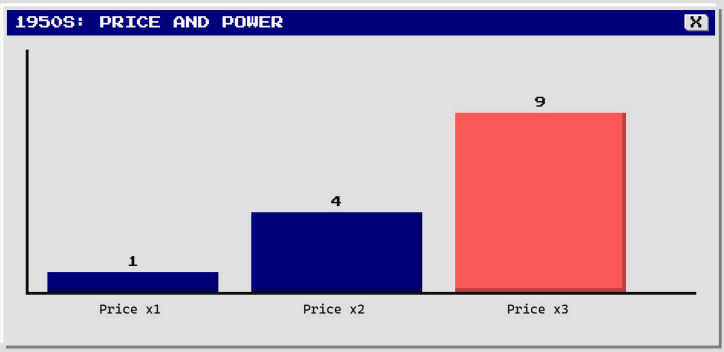
— Arthur Rock, *1990s, attributed to Arthur Rock, investor and Intel board member; often called Moore's second law (attribution uncertain between the two)*

Moore's Law is the romantic part: ever smaller, ever more powerful chips. Rock's Law is the part with the receipt. To make tiny chips you need enormous factories, and every four years the factory costs twice as much. Today we're at figures with so many zeros they look like an IPv6 address.

It's the same principle as my printer. The printer costs forty euros. The cartridges to print with it for a year cost as much as a small hatchback. Miniaturisation is cheap only for the buyer, never for the manufacturer, and hardly ever for whoever refills it.

I tried opening a silicon foundry in the garage. I have an oven, some sand and plenty of determination. So far I've only produced a brick that shows the wrong time.

GROSCH'S LAW



Spend double, go four times faster. One giant electronic brain was the best deal.

A computer's performance grows with the square of its price: spend twice as much and you get a machine four times as powerful.

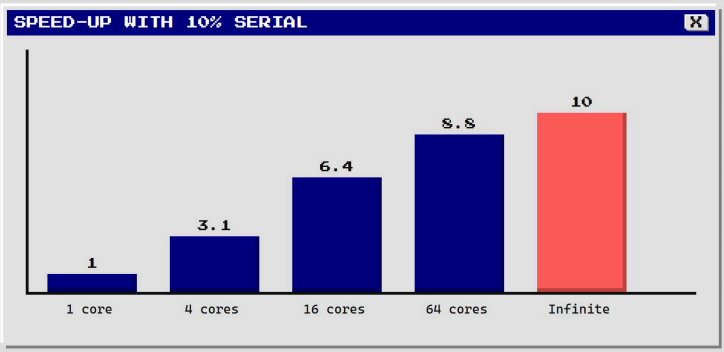
— Herb Grosch, *Formulated around 1950–1953 (sources give different years)*

In the 1950s the rule was clear: a computer that costs twice as much runs four times faster. So it made sense to buy just one, gigantic, and have everyone queue in front of it. It was the era of the corporate electronic brain, tended by priests in white coats and consulted by appointment.

Then microprocessors arrived and the law was turned on its head: today a hundred cheap machines beat one expensive monster, and so the cloud was born. Which, come to think of it, is a gigantic electronic brain we all queue in front of. History doesn't repeat itself, it refactors.

I tried applying Grosch's Law to my laptop. I spent twice as much on the version with the shinier apple. It runs the same, but now I'm scared to take it out of the house.

AMDAHL'S LAW



With ten per cent sequential, the maximum is ten times. Not eleven. Ten.

Adding processors to a job speeds up only the part that can be divided: the part that must run in sequence sets a limit that no hardware can exceed.

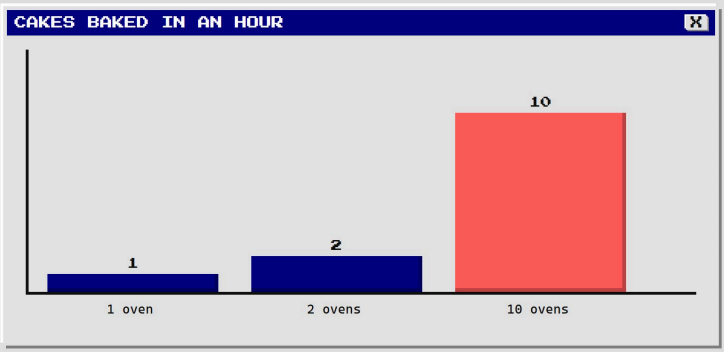
— Gene Amdahl, *AFIPS Spring Joint Computer Conference, 1967*

If nine people can't have a baby in a month, sixty-four cores can't install Windows in a second. Part of the work can be shared out. The other part, the one waiting for the previous result, stays in the queue, alone, like the post office with only one counter open.

Amdahl put it into a formula: if ten per cent of the program is sequential, you can buy every processor on the planet and you'll go at most ten times faster. Not eleven. Ten. The rest is invoice.

At my office we apply the law to meetings. We can invite twenty people to decide quickly. But in the end there's always one boss who has to say yes, and that boss is on holiday. Our sequential part even has a tan.

GUSTAFSON'S LAW



Same hour, ten cakes instead of one. The baker is happy, the scales less so.

With more processors you don't solve the same problem faster: you solve a bigger problem in the same time. So parallel computing pays off more than Amdahl predicts.

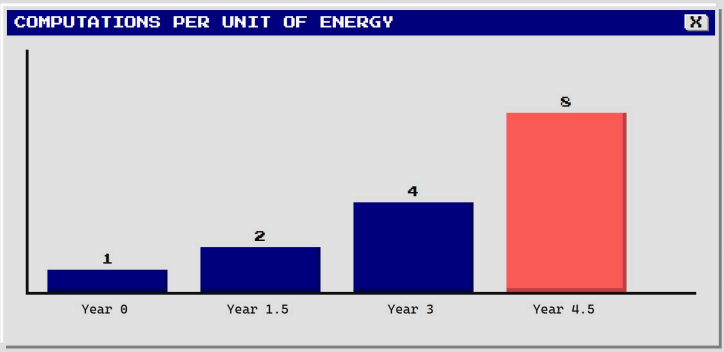
— John L. Gustafson, *'Reevaluating Amdahl's Law'*, *Communications of the ACM*, *1988*

Amdahl said: there's a limit, accept it. Gustafson, twenty years later, replied: fine, then let's make the problem bigger. If I can't bake the same cake faster with ten ovens, I'll bake ten cakes in the same time. The baker is happy, the bathroom scales less so.

In computing that's exactly what happens. Nobody uses a supercomputer to calculate yesterday's weather forecast faster. They use it to calculate tomorrow's with four times the detail, neighbourhood by neighbourhood, cloud by cloud.

I applied Gustafson's Law to my new PC. It was twice as fast. I opened twice as many browser tabs. Now it runs exactly as before, but with twice as many things I'll never read. This, ladies and gentlemen, is called scalability.

KOOMEY'S LAW



For decades, double the sums on the same power. Your phone beats a 1970s computing centre.

The number of computations a computer performs per unit of energy consumed doubles at regular intervals: roughly every eighteen months for decades, more slowly after 2000.

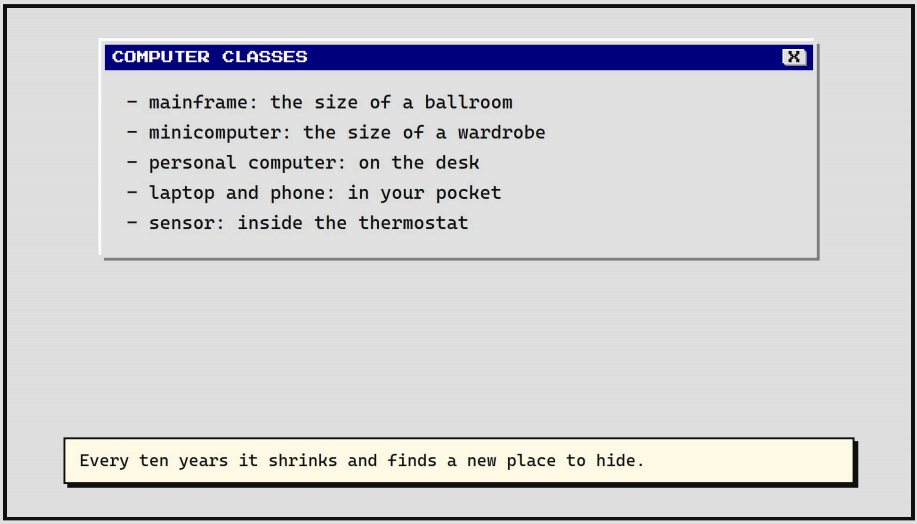
— Jonathan Koomey, *IEEE Annals of the History of Computing, 2011 (with other authors; study circulated in 2010)*

Koomey's Law is Moore's Law's environmentalist sister. It doesn't care how powerful the computer is, it cares how much electricity it drinks per calculation. And the answer, for half a century, has been: less and less. Today's phone does more sums than a 1970s computing centre and makes do with a pocket-sized battery.

It's why we have laptops, smartwatches and sensors that last for years on a button cell. Without Koomey we'd still have computers with exhaust pipes.

Unfortunately humanity reacted as usual: if each calculation costs less energy, let's do infinitely more calculations. Now we have artificial intelligences that consume as much as a small town to write a poem about a cat. Extremely efficient. Per cat.

BELL'S LAW OF COMPUTER CLASSES



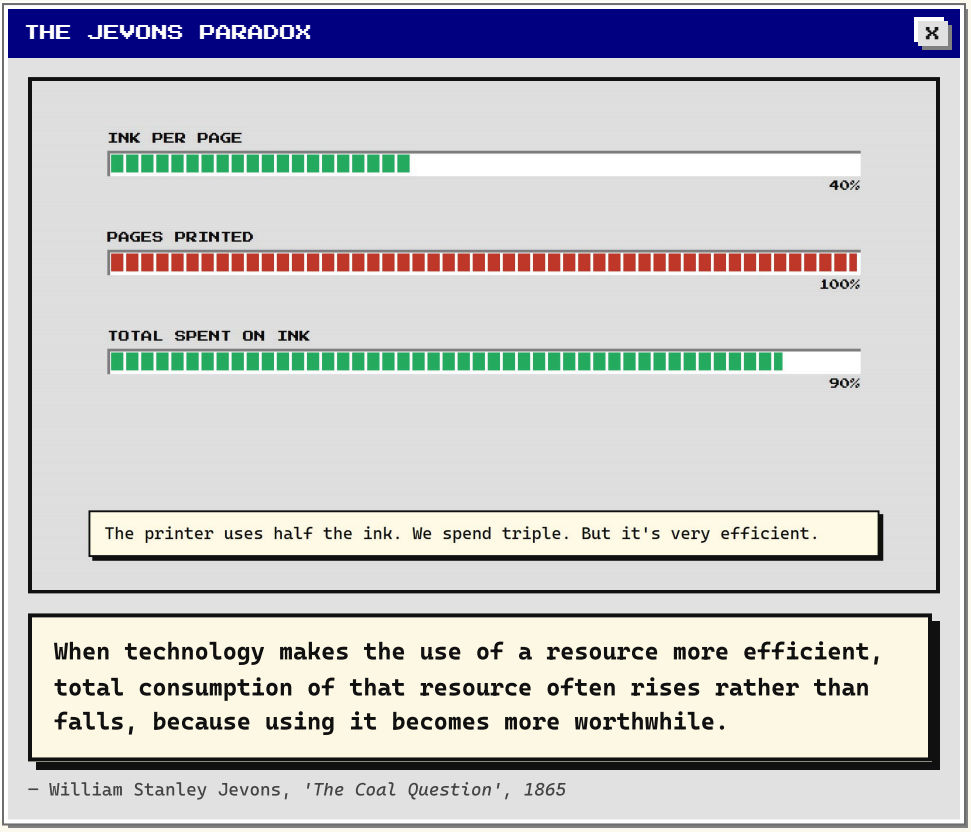
Roughly every decade a new class of computer is born, smaller and cheaper, creating a new market and a new way of using computers.

— Gordon Bell, *Formulated around 1972; described in 'Bell's Law for the Birth and Death of Computer Classes', Communications of the ACM, 2008*

First there were mainframes, as big as a ballroom. Then minicomputers, as big as a wardrobe. Then personal computers, then laptops, then phones, then watches, then sensors in the thermostat. Every ten years the computer shrinks and finds a new place to hide.

Bell's Law says exactly this: each new class costs less, does something new and brings a whole industry with it. The old class doesn't die straight away. It retires to a corner, like the mainframe that still, somewhere, runs the pensions today.

Following the trend, the next computer will fit in a jacket button. I'm ready. I've already solved the main problem: I can never find the charger, so I'm in training.

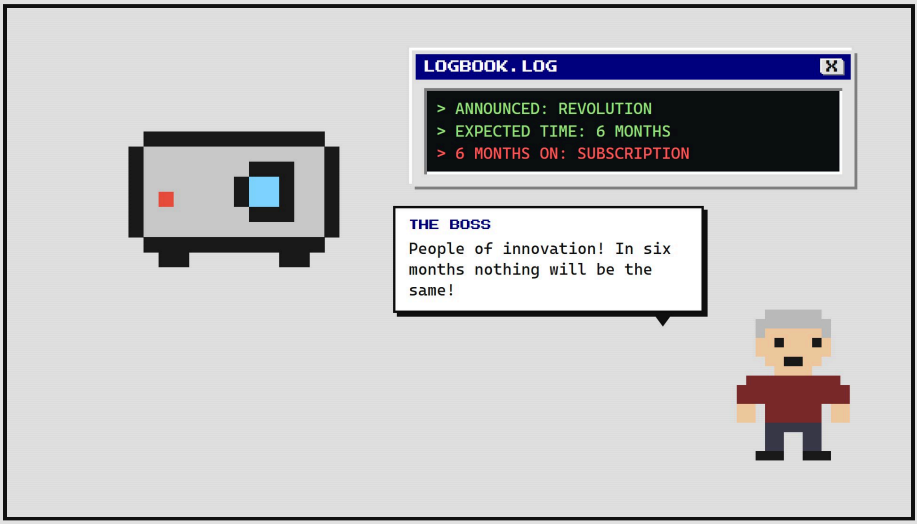


In the nineteenth century an economist noticed that more efficient steam engines didn't save coal. They made people burn more of it, because suddenly it paid to use them everywhere. The saving per unit became waste overall.

Computing is the kingdom of Jevons. Disks cost less, and we hoard terabytes of blurry photos. Bandwidth costs less, and websites weigh as much as encyclopaedias. Processors use less energy per calculation, and we fill warehouses with servers to run artificial intelligences that suggest the subject lines of our emails.

I see it at home too. I bought a printer that uses half the ink. My wife started printing recipes, train tickets, reminders for the fridge. We spend three times as much. But, she says, it's a very efficient printer.

AMARA'S LAW



We tend to overestimate the effect of a technology in the short run and underestimate it in the long run.

— Roy Amara, *Institute for the Future, 1960s–70s (often dated 1978; exact date uncertain)*

Every new technology arrives with a rally. People of innovation, in six months nothing will ever be the same again! Six months go by, and everything is exactly the same as before, just with one more subscription. So everyone says it was a bubble. And right there, quietly, the technology starts to change everything for real.

It happened with the Internet: at the end of the 1990s it was supposed to revolutionise the world in a quarter. It lost the quarter and won the twenty years. It's happening now with artificial intelligence, between those who see it as a deity and those who see it as a chatty spellchecker.

Amara's Law teaches patience. I apply it to the novelties clients bring me with great excitement. I nod, take notes and wait ten years. They were often right. It's just that in the meantime they've changed their minds twice.

COROLLARY OF THE IMAGINARY FUNCTION

THE AI'S SOLUTION

```
// clean and elegant solution
orders = load_orders()
orders.fix_everything_itself()
save(orders)

// politely commented
```

Clean, tidy, polite. It uses a command invented thirty seconds ago.

When you ask an artificial intelligence for help, the most elegant solution it offers will use a function that does not exist.

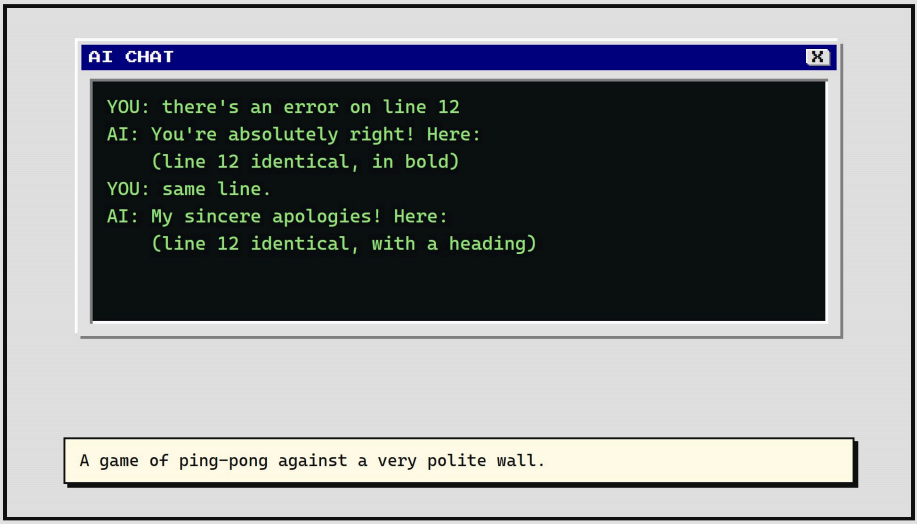
— Silvano De Poli, *previously unpublished corollary*

Generative artificial intelligence writes code with enviable confidence. It offers a clean, tidy solution, graciously commented. There is just one detail: it relies on a command nobody has ever invented. Except the AI, at that moment.

It's like asking directions from a very kind passer-by who doesn't want to disappoint you. 'The station? Straight on, second right, Garden Lane.' There is no Garden Lane. But he described it so well you almost feel sorry for the council.

Experts call these inventions hallucinations. I call them wished-for features: the AI isn't wrong, it is anticipating the programmers of the future. Meanwhile, before using any suggested line, I check that the command really exists. It's an ancient, almost artisanal practice. It's called reading the documentation, preferably for the right version.

COROLLARY OF ARTIFICIAL APOLOGIES



When you point out an error to an artificial intelligence, it will apologise with great sincerity and give it back to you identical, with better formatting.

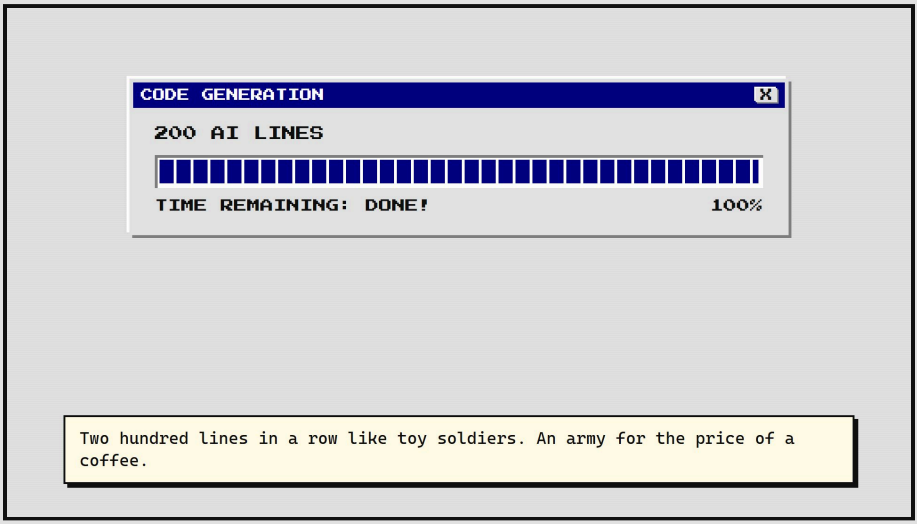
— Silvano De Poli, *previously unpublished corollary*

AI has impeccable manners. You point out a problem and it replies: 'You're absolutely right, I apologise for the error. Here is the corrected version.' What follows is exactly the same version as before, but with bold subheadings.

It's a dialogue that can go on for ever, like a game of table tennis against a very courteous wall. You insist, it apologises. You lose patience, it thanks you for your patience. In the end you're exhausted and it's still fresh, contrite and wrong.

I've learnt that the solution isn't to raise your voice but to change the question. Explain the problem from another angle, give an example, start a new conversation. With machines as with certain public offices: there's no point insisting at the same counter. You have to change queues.

COROLLARY OF THE THREE SECONDS



Code that artificial intelligence writes in three seconds takes three days to understand and a week to fix.

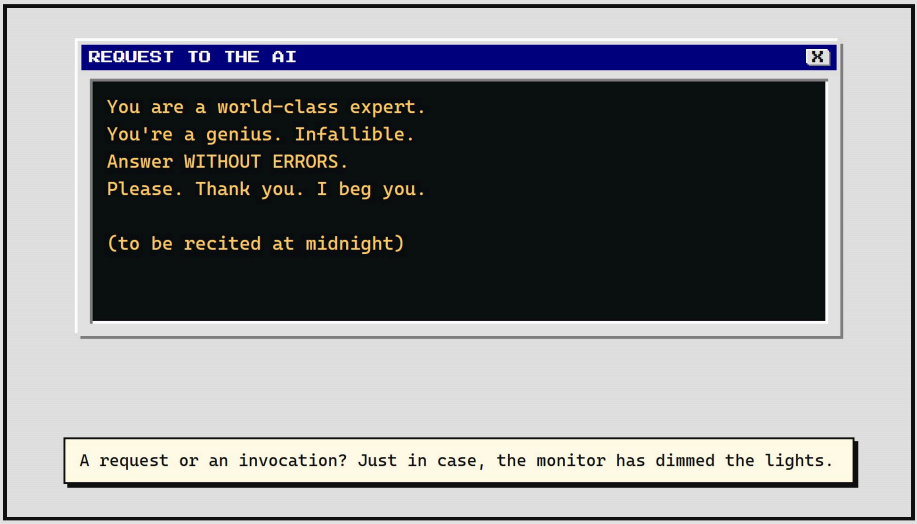
— Silvano De Poli, *previously unpublished corollary*

The speed of AI is hypnotic. You make a request and, in the blink of an eye, two hundred lines scroll across the screen, lined up like toy soldiers. You feel as though you've hired an army of tireless programmers for the price of a coffee.

Then they have to go into production. And that's when you discover those lines have to be read, one by one, like a contract drafted by a very talkative lawyer. Every function does almost what's needed. Almost is the most expensive word in computing.

The industry guru announces from the stage that programmers are finished. Meanwhile, at the office, I'm more of a programmer than ever: only now, instead of writing, I read. It's like going from chef to food critic in a kitchen that churns out dishes non-stop. Everything looks lovely. You have to taste it.

COROLLARY OF THE SOLEMN PROMPT



Writing 'without errors' in a request to an artificial intelligence does not eliminate errors: it merely makes them more polite.

— Silvano De Poli, *previously unpublished corollary*

There is a school of thought, almost a religion, according to which the secret of AI lies in the right formulas. People write enormously long requests with phrases like 'you are a world-class expert' and 'answer without errors'. They sound like incantations to be recited at midnight in front of the monitor.

The AI, flattered, replies in the tone of a professor emeritus. The errors remain, but they arrive in formal dress, with a tie and a reassuring preamble. They become harder to spot, which is worse.

A good request is actually like a good phone call to the technician: clear, with context, with an example of what happens and what should happen. No magic formulas. I tried it: I wrote 'please, be a genius'. The AI replied 'certainly'. And then got it wrong with great genius.

COROLLARY OF THE REST UNCHANGED

FULL_CODE.TXT

```
start_program()
load_data()

// the rest remains unchanged here

close_program()
```

Perfect start, perfect end. In the middle, the hard part has turned into a note.

When artificial intelligence hands you the 'complete code', the part you needed has been replaced by a comment saying 'rest of the code unchanged'.

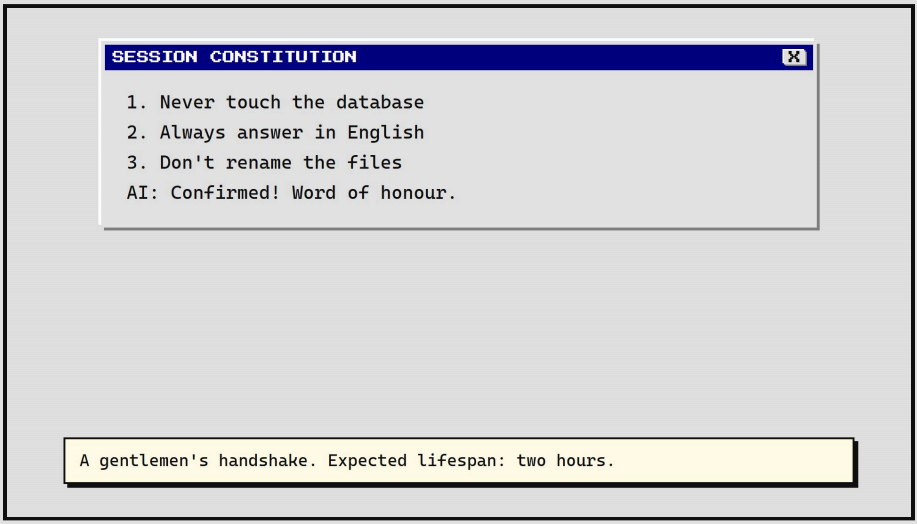
— Silvano De Poli, *previously unpublished corollary*

You ask for the whole file, updated, ready to paste. The AI replies enthusiastically: 'Here is the complete code.' You scroll. Perfect beginning, perfect end. In the middle, at the exact spot where the difficult change was, a single line: '// the rest remains unchanged'.

It's the computing equivalent of the plumber who fixes your bathroom, hands you the invoice and leaves a note on the broken pipe: 'from here on, as before'. Technically he didn't lie. Practically, the water's coming out.

The best part is that if you paste everything without looking, that comment becomes part of the program. And the program, obediently, executes nothing with great precision. Now I always read the middle bits first. As with detective novels, the culprit is never at the beginning or the end: he hides in the boring chapter.

COROLLARY OF THE FORGOTTEN RULE



In a long conversation with an artificial intelligence, the first instruction to be forgotten is the one you had declared fundamental.

— Silvano De Poli, *previously unpublished corollary*

At the start of the session the rules are laid down with the solemnity of a constitution: 'Never touch the database. Always use British English. Don't change the file names.' The AI confirms enthusiastically. It feels like a handshake between gentlemen.

Two hours and forty messages later, the constitution has become a distant legend. The AI renames the files with great creativity, replies in American spelling and suggests, with a digital smile, 'simplifying' the database by deleting a table.

It isn't malice: it has a limited working memory, like someone who goes to the supermarket without a list. Things said at the beginning fade under the weight of things said later. My solution is an old sysadmin's one: repeat the important rules, often, patiently. It works with AI. With clients, almost.

THE SYSADMIN'S ADVICE

Artificial intelligence is a formidable helper, as long as you treat it like a very brilliant and very self-confident intern. My rules, born in the field:

- **No confidential data in the chat.** Passwords, access keys, clients' personal data: they don't get pasted, not even for convenience.
- **Small steps.** One clear request at a time, with context and an example. Giant requests produce giant answers, where errors hide better.
- **Always verify.** Every suggested command or option must be checked in the official documentation, for the version you actually use.
- **Try it on a copy.** Proposed code is tested in a staging environment, never directly on the live site.
- **Ask for the tests too.** Have it write the checks as well as the code: they are the fastest way to find out whether the elegant solution works.

Used like this, it saves hours. Used blindly, it gives the hours back with interest.

NOTICE

This page has been intentionally left blank.

This is a Murphy's law too: the blank page always turns up where it isn't needed.

BONUS

HOUSE COROLLARIES

The house corollaries

After 133 laws by other people, here are mine. They were born in the field, among offices, meeting rooms and other people's desks, and until now I had only muttered them under my breath. I call them house corollaries: unpublished, certified by no authority, but tested more times than I would have liked.

COROLLARY OF THE REPLACED PRINTER

The day the new printer arrives, the old one starts working perfectly again.

— Silvano De Poli, *house corollary*

For two years it chewed paper, forgot the Wi-Fi and asked for toner that was already there. Then the courier rings with the new box, and it prints a flawless page, then another, then a colour brochure. It isn't gratitude. It's union negotiation. House advice: don't keep it ‘as a spare’. A spare printer is just a printer waiting for its moment.

COROLLARY OF THE GLOWING KEY

> **When a user says on the phone that ‘nothing works any more’, there is a good chance Caps Lock is on.**

— Silvano De Poli, *house corollary*

The password is right, the system rejects it, the user types it again harder and harder, as if the computer were hard of hearing. On the third try the account locks and the desperate call begins. I only ask: ‘Can you see a little light on the keyboard?’ Silence. Click. Thanks. It also applies to Num Lock, which turns the numeric keypad into arrows and the accounts into a mystery.

COROLLARY OF ‘CAN YOU HEAR ME?’

The first quarter of an hour of every online meeting is devoted to establishing who can hear whom.

— Silvano De Poli, *house corollary*

You connect on time, smile at the camera and the liturgy begins. ‘Can you hear me?’ ‘We can hear you but we can't see you.’ ‘And now?’ ‘Now we can see you but we can't hear you.’ Someone has an echo, someone is talking from a tunnel, one is using the monitor's microphone without knowing it. When everything finally works, the meeting is over. House advice: a headset with a microphone and a sound check five minutes before.

COROLLARY OF THE QUICK THING

Every request the client calls ‘a quick thing’ requires updating at least three things that have nothing to do with it.

— Silvano De Poli, *house corollary*

‘You just need to change a phone number on the website.’ Right. But nobody has opened the admin panel in years, the panel asks for an update, the update wants a new version of PHP, and the new version of PHP doesn't agree with a seven-year-old module. Four hours later the number has changed. The client says thanks: ‘See? It was a quick thing.’ It was. For the number.

COROLLARY OF THE ORPHAN EXTENSION

The Joomla update usually goes smoothly; what decides how the evening ends is the extension installed years ago by a developer who no longer answers.

— Silvano De Poli, *house corollary*

The core of the system is looked after by a careful community, with release notes and procedures. Then there's that module picked up long ago to show the weather on the home page. Its author has changed jobs, maybe continents. The module is still there, waiting for the update to get itself noticed. House advice: before updating, list your extensions and ask yourself who still maintains them.

COROLLARY OF TWO-STEP VERIFICATION

The verification code always arrives on the phone you left in the car, or on the one you have just replaced.

— Silvano De Poli, *house corollary*

Two-step verification is a blessing: even if they steal the password, without the phone they can't get in. Unfortunately, without the phone, neither can you. The code goes out and vibrates in a car parked two kilometres away, or on an old phone already given to a nephew. House advice: when you turn it on, save the recovery codes and keep them somewhere safe. Not on the phone.

COROLLARY OF THE CONNECTED BACKUP

> **A backup the computer can always see will sooner or later be seen by the virus too.**

— Silvano De Poli, *house corollary*

The always-plugged-in external disk is very handy: it copies everything every night without bothering anyone. Then a piece of malware arrives that encrypts the files and demands a ransom, and finds the disk right there, connected and obliging. It encrypts that too, with the same care. The backup did its duty: it copied the disaster. House advice: at least one copy must be unplugged, or kept in a versioned service that nobody can modify.

COROLLARY OF THE EXPIRED DOMAIN

> **The domain always expires when the linked credit card expired the month before and the reminders go to a mailbox nobody reads any more.**

— Silvano De Poli, *house corollary*

For years the renewal is automatic, invisible, perfect. Meanwhile the card changes, the person who registered the domain changes jobs and their mailbox keeps collecting reminders into the void. One Monday the website disappears, and the email with it. House advice: put domain and certificate expiry dates in the calendar, and have the reminders sent to a shared company address, not a personal one.

COROLLARY OF THE LOST SCANNER

> **Scanning to the shared folder stops working the day after someone changed a password, usually yours.**

— Silvano De Poli, *house corollary*

The multifunction printer saves documents to a server folder using a username and password entered years ago, in a menu reached with fourteen taps on a screen the size of a bus ticket. Change that password for security and the multifunction, with great dignity, stops delivering. House advice: give machines a dedicated account, written down in the documentation.

COROLLARY OF THE ATTACHMENT

The email that says ‘please find the file attached’ goes out without the attachment; the one with the wrong file, on the other hand, goes out first time.

— Silvano De Poli, *house corollary*

You write the message carefully, reread it, press send with satisfaction. Two minutes later the reply arrives: ‘The attachment is missing.’ The second email, written in a hurry to fix it, carries another client's quote. House advice: attach the file before writing the text, and keep on the forgotten-attachment warning that many email programs offer.

FAREWELL

We have reached the end, and nothing has broken. Nothing I know of, anyway. If you are reading the printed version, the printer has probably already expressed an opinion on the matter.

I owe you a confession. I collected these laws laughing, but many reminded me of a precise day: a server down, a client waiting, a cable that wouldn't reach. Murphy is not a pessimist. He is a colleague who has seen it all and warns you with a joke, because he knows you wouldn't listen to a sermon.

If there is a moral, it is this: things do go wrong, yes, but almost never in an original way. Tomorrow's fault is a relative of yesterday's. Whoever recognises it has already done half the work. The other half is done by a verified backup, a summary email, a notebook and the patience to check the plug before blaming the satellites.

The collection on the website stays open, like everything to do with computing. If you know a law that is missing here, keep it carefully: it means it happened to you, and so you are one of us.

Thank you for reading to the last page. Now go and make that backup. Yes, that one.

COLOPHON

MURPHY.EXE by Silvano De Poli, 2026 edition.

Set in Press Start 2P for the headings and Cascadia Mono for the text, both under the free SIL Open Font License. Laid out and published with Joomla and a bit of hand-written code, tested more times than necessary, as chapter five recommends.

Text © 2026 Silvano De Poli, depolisilvano.it. All rights reserved, except for short quotations with the source credited. Laws with a proper name belong to the tradition of their respective authors; comments, advice and house corollaries are original.

No printer was mistreated during production. One was given a dirty look.

README.TXT

If anything can go wrong, it will. In computing it happens on a Friday at six, in front of the client, with the printer that has sniffed the hurry. This book collects 133 Murphy's laws of the digital world: the classics, the ones named after a famous engineer and the ones born from years of tech support. Each is explained in plain words and annotated with the irony of someone who has spent a lifetime under desks looking for a screw. Every chapter holds one piece of real advice, the kind that saves a day. At the end, the house corollaries. You will not avoid the failures. But at least you will know their names.

CONTAINS

- [x] 133 Murphy's laws, annotated
- [x] 8 chapters, 8 real tips from the sysadmin
- [x] 10 unpublished house corollaries
- [x] 133 pixel illustrations
- [ ] guarantee that nothing breaks

FREE. LIKE FAILURES.



DEPOLISILVANO.IT