Backseat Software

January 18, 2026

What if your car worked like so many apps? You're driving somewhere important… maybe running a little bit late. A few minutes into the drive, your car pulls over to the side of the road and asks:

"How are you enjoying your drive so far?"

Annoyed by the interruption, and even more behind schedule, you dismiss the prompt and merge back into traffic.

A minute later it does it again.

"Did you know I have a new feature? Tap here to learn more."

It blocks your speedometer with an overlay tutorial about the turn signal. It highlights the wiper controls and refuses to go away until you demonstrate mastery.

Ridiculous, of course.

And yet, this is how a lot of modern software behaves. Not because it's broken, but because we've normalized an interruption model that would be unacceptable almost anywhere else.

I've started to think of this as **backseat software**: the slow shift from software as a tool you operate to software as a channel that operates on you. Once a product learns it can talk back, it's remarkably hard to keep it quiet.

This post is about how we got here. Not overnight, but slowly. One reasonable step at a time.

Software Came on Disks

There was a time when software shipped on physical media: floppy disks, CD-ROMs, sometimes even with a spiral-bound manual.

Software felt like a product back then. You bought it, installed it, and used it. If you upgraded, it was because you chose to. The software didn't constantly change underneath you, and it didn't have the opportunity to ask for your attention beyond whatever UI the developers shipped on day one.

That era had real downsides. If you shipped a serious bug, you lived with it until the next release, which could be weeks or months away. If security issues were discovered, your options ranged from "mail a patch" to "good luck." In hindsight, it's amazing we survived!

But something else was true too. When you were using the software, you were alone with it.

As a software developer, if something was wrong, you found out because users told you. Sometimes loudly. Sometimes angrily. Often on message forums or during support calls.

Feedback was slower and scarcer, but it was real. It was also "expensive" in the way that matters. You had to earn it, listen to it, and interpret it.

Always Online

Then the internet arrived, and for a while it was almost entirely upside.

Software could finally be updated after it shipped. Bugs could be fixed. Security holes could be closed. Documentation got easier. Support got easier. The idea of "ship it and hope for the best" started to fade.

Microsoft's update infrastructure is a good example of the era. Updates moved from "go download this" toward automation over time, and by the early 2000s the industry was normalizing the idea that your machine could check for and apply updates regularly.

This was a genuine leap forward in quality and safety. If you’ve ever been on the receiving end of a serious bug report, you know how valuable it is to fix something *now* rather than in the next boxed release.

So far, so good.

The Back Channel

Once software could reliably connect to the internet, it no longer just received instructions. It could talk back to the company that made it.

At first, this too was mostly good. Crash reports made it easier to fix real problems, update checks were convenient, and license activation reduced some kinds of piracy. Teams could finally see patterns in failure modes instead of guessing.

Developers like me love this kind of feedback loop, and for good reason. A tool that improves over time is better than one that doesn’t.

But that back channel didn’t stay limited to “this crashed” and “there’s an update.” It expanded, quietly, because once you can send *some* data home, the next question arrives right on schedule:

“Since we’re already connected…what else can we learn?”

Everything Gets Measured

Once software could send data home, the next natural thought was:

“Can we understand how people actually use this?”

Again, that’s not an evil thought. In fact, it’s useful! Before analytics, if you wanted to understand user behavior, you had to ask people, watch them, or infer patterns from support tickets. That requires time, empathy, and effort.

Suddenly, you didn’t have to guess anymore.

Web analytics going mainstream is one of those quiet accelerants. Google’s acquisition of Urchin in 2005, and the rise of Google Analytics shortly after, helped normalize the idea that instrumentation and dashboards were simply part of building software.

Instead of arguing in a meeting about which features mattered most, you could look at usage data. Instead of guessing where people struggled, you could see drop-offs. Instead of relying on the loudest customer, you could get a broader view.

But somewhere along the way, the center of gravity shifted.

Usage data stopped being a tool for improving software and became a tool for optimizing behavior. The question quietly changed from:

“Is this good software?”

to:

“Does this increase engagement?”

And that’s when the vocabulary starts to creep in. DAU. MAU. Retention. Funnels. Stickiness. Cohorts. Conversion. Gamification. Oh my!

If you’ve worked inside a modern product organization, you’ve heard these words so often they start to feel unavoidable.

Metrics Can Be Correct and Still Be Wrong

One of the most dangerous things about analytics is that they feel objective. A chart is a chart. A number is a number. They have the aesthetic of truth.

I’ve always liked this quote by William Bruce Cameron (often misattributed to Albert Einstein)*:*

“Not everything that can be counted counts, and not everything that counts can be counted.”

Metrics don’t measure reality. They measure what your product currently makes easy.

There's a well-known warning about this, often summarized as: *when a measure becomes a target, it stops being a good measure*. It's commonly referred to as Goodhart's Law, and the broader point shows up in multiple fields, because it keeps happening to humans in systems with incentives.

When I was at Microsoft, a team wanted to remove a feature because "the analytics show that nobody uses it." If you looked at the UI, though, that feature had been moved deeper and deeper over time:

- it used to be easy to find
- then it moved into a menu
- then into a submenu
- then into a settings panel
- then behind an "advanced" section
- then it was basically invisible

Of course nobody used it!

The analytics didn't prove the feature was unwanted. The analytics proved that we buried it.

Even worse, once a metric becomes a target, people get promoted for moving it. That doesn't require anyone to be malicious. It just requires incentives and a dashboard.

Experimenting in Production

Measuring behavior changes what feels possible:

"What if we try two versions to see which one performs better?"

This is where A/B testing enters the story.

On paper, it's an engineering triumph! Instead of arguing about opinions, you can test ideas. Instead of debating copy or layout in a meeting, you can ship both and let real-world behavior decide.

But A/B testing quietly changes the role of the product team. You're no longer just building a tool and observing how it's used. You're now running experiments on people…adjusting wording, placement, timing, friction, and flow to see what moves the metric.

At that point, the product stops being a finished artifact and starts behaving like a laboratory. Every screen becomes provisional, and every interaction becomes a hypothesis. Once that mindset takes hold, it's very hard not to optimize for what moves fastest, even if it moves the wrong thing.

There's a quieter consequence here that doesn't get talked about much. When experimentation becomes the primary decision-making tool, a strong product vision becomes optional.

Not because anyone argues against *vision*, but because you don't strictly need it anymore, and because backing a chart is safer than backing an opinion. Metrics have numbers and experiments have winners. If a decision goes wrong, you can always point to the data and say, "we followed the evidence."

Over time, this can change the role of a product team where judgment slowly gives way to iteration, and taste gives way to performance. The product still evolves, but it does so without a clear sense of direction…only a sense of momentum.

Guidance Everywhere

Once experimentation becomes the default way decisions get made, changing behavior stops being theoretical and starts being procedural.

At that point, nudges aren't a new idea. They're the obvious next move. It usually starts reasonably:

"We shipped a feature. Users might not notice it."

Fair.

So you add a little callout. Then a tooltip. Then an onboarding tour. Then a "What's New" screen. Then a little survey. Then another survey, because you didn't get enough responses the first time. By the time you're done dismissing everything, the tool has already taken more time than the task itself.

If you've ever read about “choice architecture” and nudging, this will feel familiar. The modern language for it was popularized in the late 2000s, and the core idea is simple: how choices are presented changes what people do, even if nothing is technically forced.

Then product teams go one step further. Instead of just shaping choices, you can shape *timing*. Prompts start showing up in the middle of workflows because that's when the user is “most engaged.”

The industry also has a whole discipline around persuasive design and how to move someone from intention to action with prompts, friction removal, and well-timed triggers. B.J. Fogg's behavior model is one of the more cited frameworks in this space.

Some nudges are genuinely helpful. But the same machinery that helps you discover a feature can also be used to push you into something you didn't come here to do. And once the machinery exists, it gets reused.

It's also why coming back to an app after you've been away for as little as a week can feel like a game of Whac-A-Mole. Not because you forgot how to use the tool, but because the tool has been busy while you were gone. There's new tips, new tours, new “what's new” overlays, new announcements, new prompts that all want a click before you're allowed to do the thing you actually opened it for.

Push Notifications

Then the smartphone era arrived and made interruption cheaper. Once you can send push notifications, you no longer have to wait for the user to open the tool. You can tap them on the shoulder whenever you want.

Apple's push notification service arrived with iOS 3.0 in 2009, and it's hard to overstate what a shift this was for the “who initiates the interaction?” question.

Some of this is legitimate and genuinely helpful:

- messages you asked for
- alerts you configured
- reminders you chose

But we all know where it went:

- “We miss you.”
- “You haven’t finished setup.”
- “You haven’t tried this feature.”
- “Come back and see what’s new.”

All framed as helpful. All measured in engagement. And just like that, the tool starts acting less like a tool and more like a stalker.

In Defense

To be fair, not every prompt is evil, and not every notification is marketing.

Some software is genuinely complicated, and a little guidance prevents real mistakes. Some categories are basically *made of* alerts: messaging, security, banking, calendars, delivery tracking, anything where timing actually matters.

Telemetry exists because some problems can’t be found any other way. It’s often the only method that enables teams to find the weird crashes that happen on one driver version, one device model, or one edge case you’ll never reproduce in-house.

Even the “feature tour” has a defensible origin story. Users ask for improvements, teams ship them, and then users complain they didn’t know the improvements existed. In other words, the same people who hate popups also punish you when you make changes silently. If you’ve ever shipped a big UI redesign, you already know this.

So the problem isn’t that software *ever* teaches, asks, or informs. The problem is that once a company builds the machinery to do it, that machinery becomes cheap to reuse, and the incentives gradually pull it away from “help the user succeed” toward “move the metric.”

What starts as an occasional heads-up becomes a permanent layer of UI exhaust. What starts as support becomes a funnel. What starts as a reminder becomes a habit-forming system.

That’s the drift I’m talking about. Not guidance existing at all, but guidance becoming the default posture of the tool…always talking, always nudging, always taking the first turn in the conversation.

And once the tool decides it should initiate the interaction, the rest of the story is mostly mechanics.

Even the Builders Hate It

One of the most bizarre contradictions in modern software is that the people building these engagement systems don’t like them either!

Ask anyone who works on onboarding popups, feature tours, lifecycle messaging, or in-app announcements how they feel when an app interrupts them mid-flow to announce something they didn’t ask for. The answer is almost always the same.

They hate it! Or at least they’re annoyed.

Find me the telemarketer who likes being called during their own dinner. The job exists because it works *enough* in aggregate, not because anyone enjoys being on either end of it.

So why does it keep happening? Because inside companies, the incentives are clear and the measurements are easy. You can measure clicks and track whether they led to a “completion.” You can measure whether a nudge led to the next step in the funnel.

You cannot easily measure the resentment. Or the rage clicks when they smash a button to dismiss another “did you know” pop-up. You cannot easily chart the moment a user thinks, “I used to like this product, and now it feels needy.” You cannot easily quantify the slow erosion of trust.

There’s an older framing for this that I like: *in an information-rich world, attention becomes the scarce resource*. Herbert Simon wrote about this dynamic in 1971, long before push notifications, app stores, or social media feeds.

If your business runs on attention, and attention is scarce, then the pressure to “capture” it becomes constant.

Tools Are Supposed to Get Out of the Way

As the marketing adage goes:

People don’t want a drill. They want a hole in the wall.

The drill is just the tool. The outcome is the job. Nobody wakes up and says, “I’d like to buy a new drill today!” Well, except drill enthusiasts, I suppose. Likewise, nobody wakes up and says, “I’d like to buy a new app today!” In fact, your app is in the way of their objective.

I could argue that nobody wants the hole either.

What they really want is what comes *after* the hole. They want to hang photos of family and friends, souvenirs from trips, and artwork that makes a room feel like home. The drill and the hole are both just steps along the way.

That distance matters. The further a tool is from the real human outcome, the more invisible it should be. The drill doesn’t ask how you’re enjoying your experience drilling. It doesn’t upsell you on premium hole-making. It exists to disappear the moment it’s done its job.

This is a useful way to think about software. Most users don’t want “software.” They want the outcome:

- write the document
- edit the photo
- pay the invoice
- file the taxes
- ship the code
- communicate with the team

Great tools get out of the way so the user can accomplish their goal.

Your favorite products feel like they’re not there. You open them, do the thing you came to do, and close them again without ever feeling managed, marketed to, or delayed.

Your least favorite products tend to do the opposite. You use them because you have to, not because you want to.

Everything Gets “Smart”

This pattern is spreading because “smart” is spreading. Smart TVs. Smart speakers. Smart thermostats. Smart appliances. Anything that joins your Wi-Fi can:

- update itself (often good)
- send diagnostics (often good)
- collect usage data (sometimes defensible)
- interrupt you (almost always annoying)
- market to you (almost never what you bought it for)

It’s the same story as software, just with plastic and a power cord.

One more backchannel: some “smart” TVs use Automatic Content Recognition (ACR) to identify what’s on the screen and turn that into data. It’s basically a pixel-sampled fingerprint of anything that shows up on your screen whether streamed, broadcast, or just played back locally.

If you want a more academic version of how “data collection leads to prediction which leads to intervention” becomes a business model, this is adjacent to what Shoshana Zuboff describes as surveillance capitalism. It’s not just observing behavior, but intervening to shape it.

How We Got Here

None of these events ruined software by themselves. They just made the next step easier.

- 1990s: consumer internet connectivity becomes mainstream, and “online” stops being a special mode.
- 1996–1997: PointCast popularizes “push” to the desktop (including ads): software starts initiating the interaction.

- 1997: pop-up ads arrive and interruption becomes a business model.
- 1997: Office 97 ships with “Clippy” the Office Assistant, the friendly ancestor of in-app nudges.
- 2000: “automatic updates” becomes a normal expectation for consumer operating systems.
- 2005: Urchin → Google Analytics: instrumentation and dashboards go mainstream.
- July 10, 2008: the App Store launches and app distribution becomes frictionless.
- June 17, 2009: push notifications arrive at scale on iOS (even if they weren’t first): the app no longer has to wait for you to open it.
- Nov 4, 2009: Apple announces 2 billion push notifications already delivered, early proof that “tap on the shoulder” scales.
- 2010s: “growth hacking” becomes a discipline; nudges, tours, overlays, and lifecycle messaging become standard product surface area.
- By 2011, Apple’s review rules explicitly forbade using push for “advertising, promotions, or direct marketing.”
- Mar 4, 2020: Apple changes course and allows marketing push, but only with explicit opt-in.
- 2020s: “enshittification” becomes a word people recognize because enough people feel the pattern.

People use “enshittification” to describe platform decay. What I’m describing here is one of the mechanisms that makes that decay feel personal. It’s the constant conversion of your attention into a KPI.

Designing for Quiet

I don’t want to go back to floppy disks. I like fast updates. I like security patches. I like sync. I like crash reports when they help fix real issues.

What I want is for “phone home” to be treated like a privileged capability, not an assumed right. In other domains, we treat privileged capabilities with care. We put them behind intentional choices. We build guardrails. And we treat abuse as a bug, not a growth opportunity.

Here are a few practical ways out. And yes, we’ve heard many of these before.

1) Make interruptions opt-in, and make opt-out permanent

If you want to announce a feature, fine. Put it somewhere predictable.

If you want to educate, fine. Let me ask for help.

If you want to survey me, fine. Ask at a sensible moment and accept “no” as a real answer.

Most importantly, if I turn something off, it should stay off! A tool should not require me to keep saying “not now.” Or conveniently “forget” my choices in its next update.

2) Separate product health telemetry from growth telemetry

Crash reports, performance metrics, and error logs are about stability.

Engagement nudges are about behavior.

When those get mixed together, the growth incentives win, because they produce the cleanest charts and the easiest wins. If you can’t draw a clear line between “this helps us fix bugs” and “this helps us juice numbers,” the product will drift toward the numbers.

3) Use analytics as a flashlight, not a steering wheel

Analytics are useful for asking better questions. They are not answers by themselves. Before removing a feature because “nobody uses it,” ask:

- Is it discoverable?
- Did we move it?
- Did we rename it?

- Is it used rarely because it solves rare but important problems?
- Is it used by a small set of power users who keep the whole system running?

Then talk to humans. Analytics can reduce guessing, but it can also create false certainty.

4) Optimize for trust, not just return visits

Short-term engagement can be increased by annoyance. Long-term loyalty is harder and more valuable.

The best products I use don’t constantly remind me to use them. They quietly do their job so well that I come back when I need them. That’s what tools are supposed to do.

5) Ship a real “quiet mode”

Not “quiet except for what we care about.”

Quiet.

No popups. No tours. No surveys. No “news.” No nudges.

If the product is genuinely valuable, quiet mode should improve retention, because it respects the user’s attention and intent.

Also, it’s a nice forcing function. If your product can’t stand on its own without constantly poking the user, that’s a signal. Maybe not the signal you want, but definitely a signal.

Software didn’t break all at once. It eroded slowly, one reasonable justification at a time.

Each step made sense in isolation, and each step could be defended. Together, they reshaped the priorities of an entire industry. Once software became measurable in this way, it became optimizable in this way. And optimization has a way of eating everything else.

Instead, let’s make software that respects your attention, does its job well, and lets you get on with your life. That’s what good software used to feel like and what it could feel like again. Good software is a tool that you operate, not a channel that operates on you.

As always, I love hearing from you.

Comments

5 responses to “Backseat Software”

John
January 19, 2026

Thank you for the historical perspective on how we got here. I am almost willing to pay extra money for truly quiet mode.

There’s a reason that Clippy is one of the most hated ‘features’ ever shipped by Microsoft. And I think you’ve nailed it.

Reply

John
January 29, 2026

“By 2011, Apple’s review rules explicitly forbade using push for ‘advertising, promotions, or direct marketing.”‘

…what happened to that? mainstream iOS apps spam us all the time now

Reply

Barry
January 29, 2026

Excellent. And true about more than software design. There are universal truths underneath all this behavior, instructive of how humans behave with each other generally.

My role has been the mediator between marketing, creatives, software development, and the C Suite – across several industries. Yet each company consisted of this team of critical players.

Over time, the people who rose to the C Suite came exclusively from a Marketing background. This meant that eventually, in product meetings, Marketing people dominated every discussion. The bosses and the marketers speak the same lingo. And did not understand or value the input of the other teams.

Unsurprisingly, each company as a whole came to over-valued click metrics over product development. They actively de-valued the harder to measure qualities of good UI and software design – the actual elements that customers pay money for, and judge quality with every interaction. The skills and expertise of those teams were seen as cost centers to be reduced whenever possible. While more and more got invested in Marketing programs.

Once this dynamic takes over, it's almost impossible to turn around. I watched three formerly successful companies wither into bankruptcy by strangling loyal customers with marketing plans, instead of delivering good products. As the numbers continued down, in every meeting the solution was always more Marketing (popups, emails, surveys, discount codes, etc.), because they could measure clicks, and they wanted more clicks.

Reply

J. Bee

January 29, 2026

I think this article is somewhat refusing to put it's finger on the simpler version of what's "going wrong" here. It's advertising.

All of these bad practices are a result of trying to *advertise* products and features to the end user, without their consent. Because of course, the nature of advertising is to operate explicitly *without* consent.

All this stuff happens because of the monetary incentive, and the view that users are just undifferentiated purchase bots that will probably buy more stuff if you throw it in their faces.

It’s *not* actually morally OK to advertise to people without giving them the option to refuse. It’s not OK for instance, for Apple to advertise it’s products and services within the OS as they now do, and not allow people to opt out from those advertisements.

This is exactly what made Microsoft of the 90’s and early 2000’s so absolutely vile. The fact that they extended themselves into services, and then sold their souls by advertising those services, (as well the ‘cool features’ of their own OS’s), within the OS itself. The whole system became a nightmare of pop up suggestions and requests to buy into their services. Apple is definitely moving strongly in that direction.

Switching to Apple *was* the breath of fresh air at the time, but now they’re just doing the exact same thing that we all hated Microsoft for back then.

because: money

Reply

Dmitri Zdorov

January 29, 2026

It’s not just advertisement, it’s hustling culture in general. Hustlers just do better overall. They sell more, they earn more, they get more prizes, they’re more popular. And you can say that is chasing a quick buck, short-term profits, or maybe it’s just what you get when there is too much competition, which is not as bad as too little, but still not optimal.

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment *