5 Golden Rules

of Great Software

Written By:

Mark Orttung, CEO

Nexient

Humans are tool-making creatures. Technology is what we're supposed to be good at. So why is so much of the software we use every day so bad?

You know what I'm talking about: Help boxes that don't help, buttons that don't seem to do anything and features hidden in sub-menus three levels down.

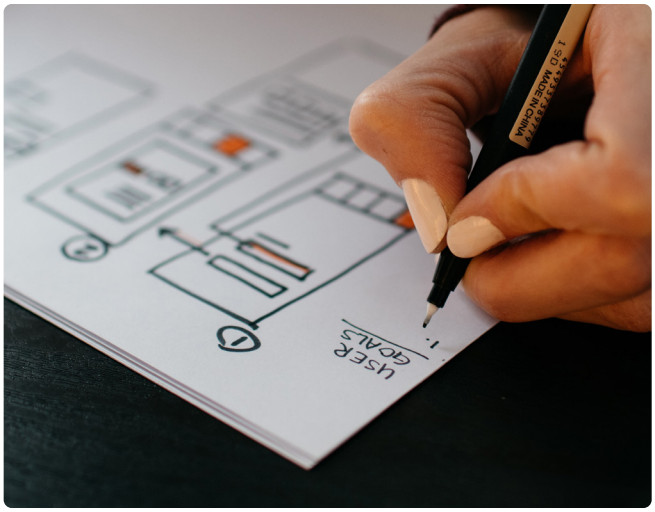
You don't have to be a coder to know when software is good or bad. You can feel it the same way you can feel the balance and heft of a well-made hammer or the flimsy construction of a cheap door handle.

Even among professional coders, the highest praise you can give software is that "it just works." Developers know how much effort it takes to create the sensation that software is an extension of your mind, presenting everything as you need it, as soon as you want it. But they know something else that's important: Making good software is hard, but it isn't complicated. It boils down to a handful of golden rules. They can be memorized in minutes, but mastering them is something you can work on your entire career.

In this series, I'll walk through five golden rules every business needs to emulate in order to avoid creating bad software.

Rule No. 1

Know Your User



Why is this the most important and hardest rule to follow?

In some ways, all the other rules are just restatements of the idea that to produce good software, you have to understand your users.

You know what I'm talking about: Help boxes that don't help, buttons that don't seem to do anything and features hidden in sub-menus three levels down,

You don't have to be a coder to know when software is good or bad. You can feel it the same way you can feel the balance and heft of a well-made hammer or the flimsy construction of a cheap door handle.

Even among professional coders, the highest praise you can give software is that "it just works." Developers know how much effort it takes to create the sensation that software is an extension of your mind, presenting everything as you need it, as soon as you want it. But they know something else that's important: Making good software is hard, but it isn't complicated. It boils down to a handful of golden rules. They can be memorized in minutes, but mastering them is something you can work on your entire career.

Engineers like to think their way to the right answer, but that doesn't work in this case. Knowing your user means entering into their experience. To do this, you have to physically go to them and observe them in action.

A colleague was working on a point-of-sale system for a well-known auto-service chain. He visited a nearby location and noticed something that changed his whole understanding of the project. Everyone was wearing gloves, even at the register, in case they needed to help install wipers or do a quick oil change in the garage. The touchscreen solution my colleague had originally envisioned couldn't possibly work. The latex of the gloves would not work with the screen. He also got new insights about the user interface that he would never have had by staying in the office. He would need large buttons, plenty of white space and a focus on enabling users to complete tasks fast.

When product teams are too far removed from users, the results are atrocious. Consider the in-car navigation system for one of America's largest auto manufacturers. On the face of it, a built-in system with a large-console touchscreen should be more convenient than a mapping application on a phone. But as anyone who has used one of these in-car systems can attest, the opposite is the truth.

It takes two seconds on Google to look up an address using speech-to-text or autocomplete, and Google automatically identifies whether you're searching for a point of interest, a residence or a business. Meanwhile, in the car, you have to painstakingly key in the address, pausing for the system to register each letter. If by some miracle you manage to enter the address correctly, you have to specify whether you're looking for a point of interest or a building before the nav system will give you directions. It's a dreadful interface that seems to have been designed by someone who didn't give the first thought to the needs of a harried parent with two cranky kids in the back seat.

As a developer, it's easy to get carried away with clever ideas that don't actually track to what users need. It's also tempting to follow a product spec to the letter, without getting real user feedback. The best way to avoid both mistakes is to observe the user up close, ask lots of questions and, above all, pay attention.

Great software companies take this idea to the next level by capturing these observations in personas. Look around their offices, and you'll often see displays with names, photos and back stories of representative users, from truck drivers to accountants. These personas remind product teams to think about their users as distinct personalities, with distinct needs. They remember to empathize rather than just ticking off requirements on a spec sheet.

I need to make special mention of corporate software here. In-house software developers are often squeezed for resources, and user research may be the first line item axed. Why bother when your employees don't have a choice of which software they'll use? This is a false economy, of course. Every time poorly designed software confuses, frustrates or delays your employees, you're paying more to get the job done.

Achieving the first rule of software development - know your user - is a serious undertaking. Empathizing with your users and anticipating their needs requires deep understanding. Once you've mastered rule No. 1, you'll be well-equipped to accomplish the rest.

Rule No. 2

Provide Consistency

Businesses don’t have one digital presence anymore - they have many.

They interact with customers over email, social media, their websites and live chat, just to name a few.

But too often, it feels like the left hand isn’t communicating with the right. Your bank emails to say you’re preapproved for a new credit card based on your spending level. But when you click the link, you’re asked for your name, address and social -- information you’ve given your bank countless times. Instead of the site pulling up the data from some central location, you have to enter it all again. You’re the customer, but you feel like you’re doing the work.

Or you call your internet company, provide your details and describe your problem to the automated system. After 20 minutes of being put on hold, you finally talk to a representative -- only to have them ask you again for your account number and the nature of your problem. If you’re unlucky enough to get passed to another service tier, you go through the process yet again, Groundhog Day-style.

Why can’t your service provider pass a few pieces of data from department to department? The answer is depressingly simple. You feel like you’re dealing with six different organizations because you basically are. Large enterprises are organized into silos that don’t talk to each other, and as a result, they can’t act with a single voice.

Under the hood, each of these customer touchpoints relies on different systems that are run out of separate organizations with their own P&Ls, tech teams and management. From their customers' point of view, however, it's just one company that can't get its act together.

Delivering a consistent customer experience across channels (an omnichannel approach) isn't easy, which is why so many organizations fail at it.

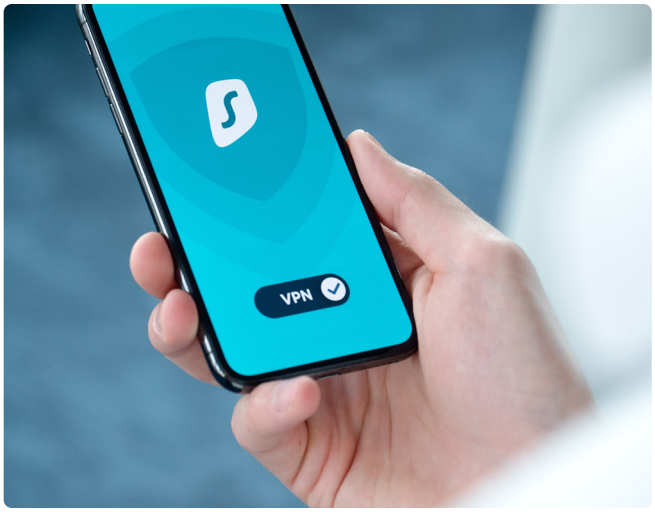
Businesses need to start by establishing common visualization (such as branding and icons) and interaction standards (such as the order in which information is presented) across touchpoints. At the same time, customer choices must remain appropriate for the specific channel (for example, using the "hamburger" version of menu options in a mobile context, or common phrases in an automated voice system or a chatbot).

The trickier part is creating logic for those inevitable scenarios when there are multiple data sources serving up conflicting information. Maybe you signed up for your checking account with one email address, but set up your mortgage loan with the same bank using a different email address. Great omnichannel companies manage this by creating business rules to coordinate data, resolving duplicate information and optimizing business resources.

Providing seamless experiences is harder than it looks. But customers don't care how hard it is. Your competitors are just a click away, so they're not inclined to cut you any slack. The only solution is to do what it takes to give them the same service in every channel, on every device.

Rule No. 3

Make It Work



It should go without saying, but software has to, you know, function.

In this era of moving fast and breaking things, this basic principle sometimes gets overlooked.

Last year, Chipotle celebrated National Avocado Day with a coupon code to add free guacamole to any order. The fast-casual Mexican food chain apparently didn't anticipate the demand from avocado-crazed Americans, who swamped Chipotle's web servers, making it impossible to redeem the offer and forcing Chipotle to extend the promotion and issue an apology committing:

> "to making our digital options bigger and better in the future."

Embarrassing, for sure. Luckily, these are the sorts of problems software engineers love to tackle since there are usually clear right answers.

Before you launch a new product or feature, consider this basic checklist:

1. **Push product performance to its limits.** How fast can you execute a single transaction? Can your system handle not just normal daily volumes, but the kind of Black Friday or Prime Day peaks that might make or break your business?

2. **Rehearse, rehearse, rehearse.** How much have you practiced the production release? Are you completely assured the rollout will work flawlessly? You can reach that confidence by automating the deployment process, making the release you've practiced as easy as pressing a button.

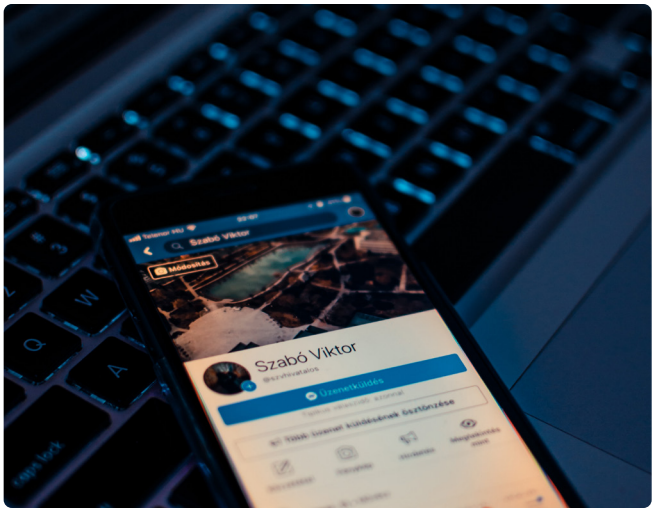
3. **Verify accessibility.** Sure, your software works perfectly in all the most common browsers and devices, but have you considered the experience for a blind person using a screen reader? How about for someone navigating with a mouthstick instead of a mouse? Even if you don't know the particular challenges a future user might face, applying the principles of accessible design can actually improve the logic, function and performance of your site for all users.

4. **Expect the unexpected.** What happens when your user enters information in an unusual format? Will they get a hopelessly cryptic error message or a helpful, maybe even humorous instruction on how to try again?

5. **Document mistakes.** Speaking of the unexpected, are you automatically tracking errors and building in better solutions over time? Anticipating what might happen after your software escapes the safety of your test lab is critical. You'll need to design it to respond gracefully to what you didn't anticipate and flag those errors for future improvements.

Rule No. 4

Be Secure (Yet Practical)



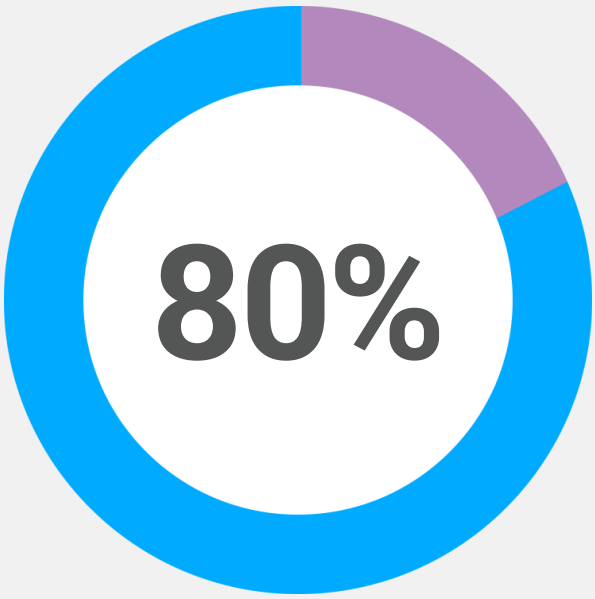
Data is digital, and digital data is vulnerable.

Personal data, corporate secrets -- it's all fair game for cybercriminals. It doesn't matter how performant or user-centric your software is if it exposes sensitive information for pilfering.

That said, you need to strike a balance. Security is not a yes-no question; rather, it's a compromise between risk and return. All security creates inconvenience. The question is whether the value of what you're trying to protect justifies the trouble. If you're designing a banking site, you can justify almost any amount of security: strong passwords, captchas, two-factor authentication. But should you ask the user to enter a two-factor code to check their gas bill? That's harder to say.

Sometimes the right move is to loosen up a little. In the early days of the internet, when most people worked on large monitors, leaving a password visible was unthinkable. Developers always made sure it was hidden behind dots as you typed. But with the advent of smartphones, obscuring passwords was often more trouble than it was worth. Tiny touchscreen keyboards made typing mistakes more common and harder to catch when users couldn't see what they had typed. At the same time, applications were demanding increasingly complicated passwords with numbers, upper and lowercase letters, and special characters, making mistakes even more likely.

Users grew frustrated, and businesses felt the pain, too. At one utility I know of, more than 80% of support calls had to do with username and password complaints. Most of the time, the customer had left the Caps Lock on or were just mistyping one character. As the number of these simple errors increased, so did support costs, giving businesses an incentive to find some middle ground.



SUPPORT CALL COMPLAINTS

The result was the now-familiar “eye” icon, which allows users to reveal the text in the password field, letting them decide how much risk they’re willing to take. They might leave the password obscured on a crowded train, but reveal it at home where the risk of snooping is lower.

There is no simple answer as to how much security an application needs. In the end, you have to be guided by what’s best for users, which once again means understanding who they are, how they’ll be using the product and what sort of balance you can strike between security and convenience.

Rule No. 5

Be Delightful

Aesthetics matter.

Software design doesn't need to be groundbreaking or conspicuously pretty, but users will notice a dated look or one where it's clear the developer doesn't know what arrangement of fonts, colors and shapes appeal to the eye.

When you get it wrong, users have a visceral reaction they probably couldn't explain if you asked them. It's the way we humans are wired to favor certain visual combinations.

You can design a brilliant navigation layout for an e-commerce site, for example, but if the colors are loud or clashing, the user will be less likely to complete the purchase. This is due to a well-documented phenomenon called the aesthetic-usability effect, in which people perceive pleasing designs to be more usable than displeasing ones. You still have to make your site usable, but users are more likely to stick around and complete the buying process if you pay attention to appearances.

Ugly design hurts more serious applications as well, diminishing users' confidence and willingness to use the software. The principle that "you catch more flies with honey" applies as well to software as it does to conversation.

> "you catch more flies with honey"

In Closing

That's all there is to it. Every delightful app you've ever used has probably followed these five rules. Every slapped-together software monstrosity you've ever encountered has likely broken a few.

The reason bad software persists is that, like anything worth doing, adhering to these rules takes work. But if software is truly eating the world, then the quality of that software matters. It's not just a question of selling more widgets. As we saw in 2018 when poor software design contributed to a nuclear panic in Hawaii, bad software can have world-changing consequences.

If you don't work for a software company, this might feel like someone else's problem. But today, just about every bank, utility, retailer, government agency and even nonprofit organization is also a software company. You depend on great software to serve your customers, employees and mission. Life is too short for crappy software. That's why it's more important than ever to remember the golden rules.

Thank You!

We’d love to hear from you:

info@nexient.com

#Nexient

#Nexient

Nexient

#JoinWhatsNext