Chapter 11
Eliminate Ambiguity

In software development, ambiguity is expensive. Ambiguous specifications lead to broken implementations, and ambiguous communication creates conflict and waste.

Once you recognize ambiguity, it's easy to fix—the hard part is recognizing it in the first place.

In this chapter, I'll show common anti-patterns that create ambiguity and how you can avoid them in your writing.

Replace ambiguous pronouns

A common way to confuse the reader is with ambiguous pronouns.

As a refresher, pronouns are words like "it," "that," "he," "she," or "they." Pronouns are a shorthand for things you've already named.

Without pronouns, writing feels repetitive and cumbersome, but pronouns can confuse the reader if you're not careful.

Here's an example:

> ✗ **BAD** My officemates put annoying wind-up toys on my desk, so I threw them out the window.

What did you throw out the window? The toys or your officemates?

Even if the reader can guess what meaning you intended, it's better to be explicit. You can rewrite the sentence to name what you threw out the window:

> ✓ **GOOD** My officemates put annoying wind-up toys on my desk, so I threw **the toys** out the window.

If repeating the word feels repetitive, keep the pronoun, but rewrite the sentence to eliminate the ambiguity:

> ✓ **GOOD** My officemates love pranking me by interfering with my workspace. Yesterday, I found a bunch of annoying wind-up toys on my desk, so I threw **them** out the window.

When you use a pronoun, ask yourself if there's any ambiguity in the pronoun's meaning.

Examples of ambiguous pronouns

> ✗ **BAD** At work, we switched from Python to Go, **which** I appreciated.

Did you appreciate Python? Or the switch away from Python?

> ✓ **GOOD** At work, we switched from Python to Go, **a change** I appreciated.

✗ **BAD** Raw pointer operations confuse junior developers, so **they** aren't allowed in our codebase.

What's forbidden in the codebase? Are the junior developers barred from accessing the codebase because the raw pointer operations would confuse them? Or is there a rule against using raw pointer operations to make the codebase safe for the junior developers?

✓ **GOOD** Raw pointer operations confuse junior developers, so we forbid **those operations** in our codebase.

✗ **BAD** The client extracts an authentication token from the JSON response. If **it**'s malformed, **it** throws an error.

Who's throwing the error? And are we talking about a malformed token or malformed JSON response?

✓ **GOOD** The client extracts an authentication token from the JSON response. If the response is malformed, the client throws an error.

Avoid long-distance pronouns

Imagine scrolling through a blog post for the information you need, and suddenly you find yourself in this section:

> ✗ **BAD**
>
> That was the critical piece for him: the final piece of the puzzle! After months of difficult work, it looked like he'd cracked it.
>
> He called Jacobs and excitedly announced that he'd solved it. She couldn't believe how sudden it all was. "It was just incredible hearing how quickly it all came together. He got it done in just two weeks, while my team had been on it for months!"

Who are these people? What problem did they solve? None of it makes sense! You have to backtrack in the document to decode the meaning of each pronoun, then scan forward again to re-read the pronoun-heavy passage.

The problem goes away if you replace the pronouns with explicit names:

> ✓ **GOOD**
>
> **The truncated log file** was the final piece of the puzzle! After months of difficult work, it looked like **Joe** had finally identified **the root cause of the kernel driver bug**.

When a pronoun refers to an idea in a prior paragraph, I call it a "long-distance pronoun."

The first problem with long-distance pronouns is skimmability. The reader can't scan through your document because long-distance pronouns force the reader backward to understand what each pronoun means.

The second issue with long-distance pronouns is that they dilute the reader's focus. Think about your experience dealing with global variables. A five-line function with only local variables is easy to understand because you can narrow your focus on just five lines. Once you throw in a global variable, the function is no longer easy to read because distant changes to the global variable can now affect the function's behavior.

Long-distance pronouns are a bit like global variables. Instead of allowing the reader to focus their attention on the current paragraph, the reader must carry around context from several paragraphs ago.

Treat paragraph and section breaks as context resets for pronouns. If you were using a pronoun in the previous paragraph, name the concept again before using the pronoun as shorthand. Be more strict about this rule when the two paragraphs are separated by an image or a heading.

✗ **BAD** Use pronouns to refer to concepts from prior paragraphs.

The VP showed me **the spec** for the project, and **it** worried me. **It** declared that we could rewrite the bank's 5-million-line COBOL backend as a React.js app in a single sprint. **It** didn't have any plan for equivalence testing beyond a single line that said "Verify no regressions."

Worse, **it** allocated only two weeks of dev time for the entire rewrite. When I asked **him** how that would be possible, **he** pointed out that React itself contains several million lines of code, so most of our COBOL backend was probably already there.

He confidently reassured me that React was "a super easy language." **His** niece had used **it** to create an impressive Minecraft website in a single afternoon. **He** offered to link **it** in the spec as a reference for how our final product should look.

It's difficult for the reader to trace the meaning across paragraphs because the pronouns straddle multiple paragraphs. Redeclaring the pronouns in each paragraph alleviates ambiguity:

✓ **GOOD** Use pronouns within paragraphs rather than across paragraphs.

The VP showed me **the spec** for the project, and **it** worried me. **It** declared that we could rewrite the bank's 5-million-line COBOL backend as a React.js app in a single sprint. **It** didn't have any plan for equivalence testing beyond a single line that said "Verify no regressions."

Worse, **the spec** allocated only two weeks of dev time for the entire rewrite. When I asked **my VP** how that would be possible, **he** pointed out that React itself contains several million lines of code, so most of our COBOL backend was probably already there.

My VP confidently reassured me that React was "a super easy language." **His** niece had used **it** to create an impressive Minecraft website in a single afternoon. **He** offered to link **her website** in the spec as a reference for how our final product should look.

Eliminate ambiguous modifiers

A "modifier" is a word or phrase that modifies the meaning of another word or phrase in the sentence.

The simplest modifier is an adjective like "smart" or "fast":

> ✓ **GOOD** Users prefer the **fast** website.

Modifiers can also be phrases:

> ✓ **GOOD** That's the website **that users prefer**.

Modifiers are all well and good until they become *ambiguous* modifiers, which are bad. Here's an example:

> ✗ **BAD** **Submerged in floodwaters**, the developers were unable to access the production server.

Are the developers underwater? Or is the server?

In that sentence, "Submerged in floodwaters" is the modifier. It's *supposed to* modify "the production server," but the sentence structure leads the reader to think it modifies "the developers," the closest noun.

To eliminate the ambiguity, reorder the sentence to place the modifier next to the noun it modifies:

> ✓ **GOOD** **Submerged in floodwaters**, the production server remained inaccessible to the developers.

Better yet, eliminate the passive voice for an even clearer sentence:

> ✓ **GOOD** The developers were unable to access the production server, as the storm had submerged it in five feet of floodwaters.

Modifiers can also create ambiguity when they modify something that's absent from the sentence:

> ✗ BAD **As a lifelong C developer**, my dog is named `#define`.

Despite how much fun it would be, the author *probably* doesn't have a programming dog.

> ✓ GOOD **As a lifelong C developer**, I named my dog `#define`.

The technical names of these errors aren't important as long as you recognize the patterns, but here they are in case you end up in a conversation with a grammar enthusiast:

- **misplaced modifier**: The modifier is in the wrong part of the sentence, misleading the reader about what it modifies.
    - "Submerged in floodwaters, the developers…" is an example of a sentence with a misplaced modifier. The modifier is next to "the developers" instead of "the production server."
- **dangling modifier**: The modifier refers to something the sentence fails to name.
    - "As a lifelong C developer, my dog…" is an example of a sentence with a dangling modifier. "As a lifelong C developer" modifies the speaker, but the sentence never mentions the speaker.

Examples of ambiguous modifiers

> ✗ BAD **Running out of memory**, the reliability engineer killed the database process.

As written, it sounds like the reliability engineer was the one running out of memory.

> ✓ GOOD **Noticing the system running out of memory**, the reliability engineer killed the database process.

✗ BAD **Buried deep within the legacy codebase**, the junior engineer finally identified the bug.

As written, it implies that the junior engineer is physically buried under a bunch of legacy code.

✓ GOOD The junior engineer finally identified the bug, **buried deep within the legacy codebase**.

Don't lead the reader down the garden path

A "garden-path sentence"[1] is one that forces the reader to go backwards and reinterpret ambiguous words.

Imagine that a sentence begins like this:

> The user logged in as the administrator…

So far, you expect this sentence to be about a user logging in with admin credentials.

But, wait! You didn't let me finish:

> The user logged in as the administrator **was taking a luxurious bubble bath**.

Initially, "as the administrator" seemed like it described how the user logged in, but "was taking a luxurious bubble bath" forced you to backtrack and reinterpret the meaning of "as the administrator."

These sentences are called "garden-path sentences" because they lead the reader "down the garden path." If you weren't around 100 years ago when "leading someone down the garden path" was cool slang, it just means to trick them.

You can identify garden-path sentences in your writing by reading your writing aloud. If a sentence makes you stumble, consider whether you've created a garden-path sentence.

Another way to catch garden-path sentences is to take some time away from your writing and come back with fresh eyes. When you find yourself surprised by the ending of a sentence, consider a rewrite to make the sentence feel more intuitive.

---

[1] https://en.wikipedia.org/wiki/Garden-path_sentence

Examples of garden-path sentences

> ✗ **BAD** The package downloaded almost ten million times from npm.org turned out to contain malware.

On first read, you probably thought that the package was performing the downloads.

> ✓ **GOOD** The package reached almost ten million downloads on npm.org before users discovered that it contained malware.

> ✗ **BAD** The unexpected method returns default to null.

On first read, you probably thought the unexpected method was returning something.

> ✓ **GOOD** The return value from the unexpected method defaults to null.

> ✗ **BAD** The performance tool records while loops parsing the massive JSON payload exhaust memory.

On first read, you probably thought the performance tool recorded `while` loops.

> ✓ **GOOD** According to telemetry that the performance tool records, the server executes loops while parsing massive JSON files and exhausts memory.

Avoid multi-verb pile-ups

Garden-path sentences have a close sibling I call "multi-verb pile-ups." Multi-verb pile-ups occur when you cram a bunch of words together that can either be verbs or non-verbs depending on the context.

Whereas garden-path sentences lead the reader down the wrong path, multi-verb pile-ups are so confusing that the reader can't find the path at all.

Here's an example:

> ✗ **BAD** Patch release lock controls bypass proxy filter limits.

How many times did you have to read that sentence until you reached a sensible interpretation? The clearer rewrite is:

> ✓ **GOOD** The system has controls that manage how it acquires locks for its patch releases. These controls bypass the normal limits that apply to the proxy filter.

The reason the first version was so challenging to parse was that every word in that sentence *can be* a verb, depending on the context. When you chain them all together, the reader has to ask each word, "Are you the verb? Or are you the verb? Or you?"

Multi-verb pile-ups can technically happen with any part of speech, though they're most noticeable with verbs because verbs drive the sentence.

Look out for chains of verbs in your writing. To reduce confusion:

- Replace ambiguous verbs with unambiguous ones.
- Insert unambiguous words between the ambiguous words to reduce potential misinterpretations.

Examples of multi-verb pile-ups

> ✗ **BAD** Configure controls to change email display format options.

> ✓ **GOOD** Configure what options are available to the user for changing the display format of their emails.

✗ **BAD** The server serialized delete calls to JSON.

✓ **GOOD** The server received several requests to delete records. It serialized those requests to JSON.

✗ **BAD** Cache store query requests block host updates.

✓ **GOOD** When the host has pending requests to the cache store, those requests prevent system updates.

Prefer subject-first sentences

Consider this simple sentence:

✓ **GOOD** Michael deployed the new app.

You can read that sentence straight through with no ambiguity. It goes subject, verb, object, which is about as basic as sentences get.

Now, consider what happens if we push the subject deeper into the sentence with modifiers:

✗ **BAD** At 2 AM on a cold December morning, feeling bleary-eyed and disoriented because he'd just awakened from a terrifying dream about floating-point arithmetic, Michael deployed the new app.

The second sentence is still grammatically correct, but the reader can't understand what's happening until the end. The sentence forces the reader to track a series of disembodied details. It's not until the word "Michael" that the reader can "pop" the stack and apply the details to a subject.

Look at how much easier the reader's job is when the subject appears first:

> ✓ **GOOD** Michael deployed the new app at 2 AM on a cold December morning, feeling bleary-eyed and disoriented because he'd just awakened from a terrifying dream about floating-point arithmetic.

The second version doesn't have as much stylistic flourish, but it's easier to understand. The reader doesn't have to push anything onto their mental stack because they can immediately apply each detail to a subject they've already identified.

Subject-first sentences match how people speak in real life. "I read an interesting article this morning," sounds like something a co-worker might say at lunch. Unless you had a particularly odd co-worker, they probably wouldn't say, "Feeling bored during the morning commute and squeezed awkwardly on my subway seat between two large men with lap dogs, I read an interesting article."

Note that I said to *prefer* subject-first sentences, not to use them exclusively. If you structure all of your sentences to begin with the subject, your writing will sound awkward and repetitive.

Sidenote: What I'm calling a "subject-first sentence" is what linguists call a "right-branching sentence."[2] Conversely, they describe sentences as "left-branching" when the modifiers appear first. The naming is too boring to explain here, but if you hear someone talk about right- or left-branching sentences, they just mean subject-first or modifier-first sentences.

[2] https://en.wikipedia.org/wiki/Right-branching_sentences_in_English

Summary

- When using pronouns like “it,” “that,” “he,” “she,” or “they,” check that the pronoun refers unambiguously to a single concept.
- Avoid using “long-distance pronouns” to refer to ideas in prior paragraphs.
    - Treat paragraph breaks as opportunities to remind users of the context for pronouns.
- When using modifier phrases, place them close in the sentence to the thing they modify so that the meaning is clear.
- A “garden-path sentence” is a sentence where the ending causes the reader to go back and reinterpret prior words.
    - If the ending of a sentence surprises you or causes you to stumble over your words when reading aloud, consider rewriting it to be less surprising.
- A “multi-verb pile-up” is when you chain together several words in succession that can act as verbs or non-verbs depending on context.
    - Notice when you’ve used a series of words that can be verbs, and look for ways to split them up.
- Place the subject early in the sentence.
    - Subject-first sentences are closer to spoken English and require the reader to juggle less information.