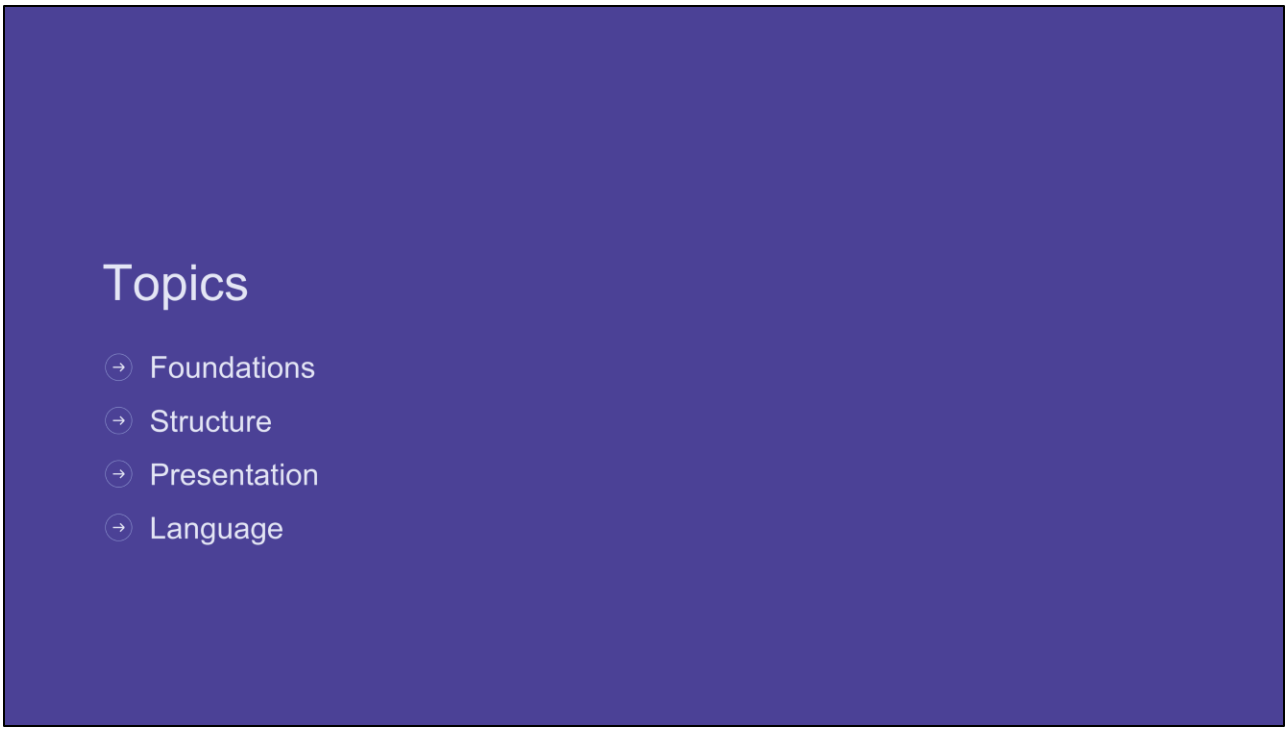
A Writing Guide for Engineers

Tips on Doc Structure, Presentation, & Language

Jim Foley
Fall 2022

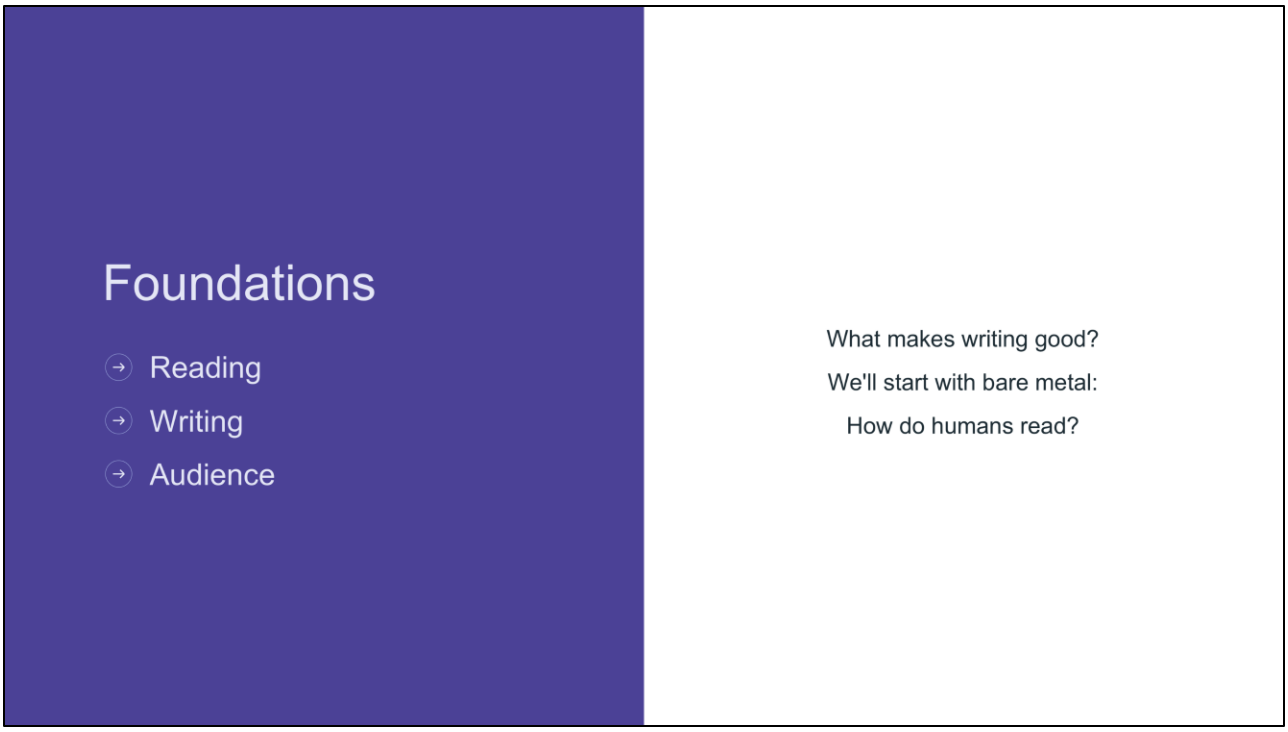
This is a talk for engineers on how to write *internal* engineering documentation.



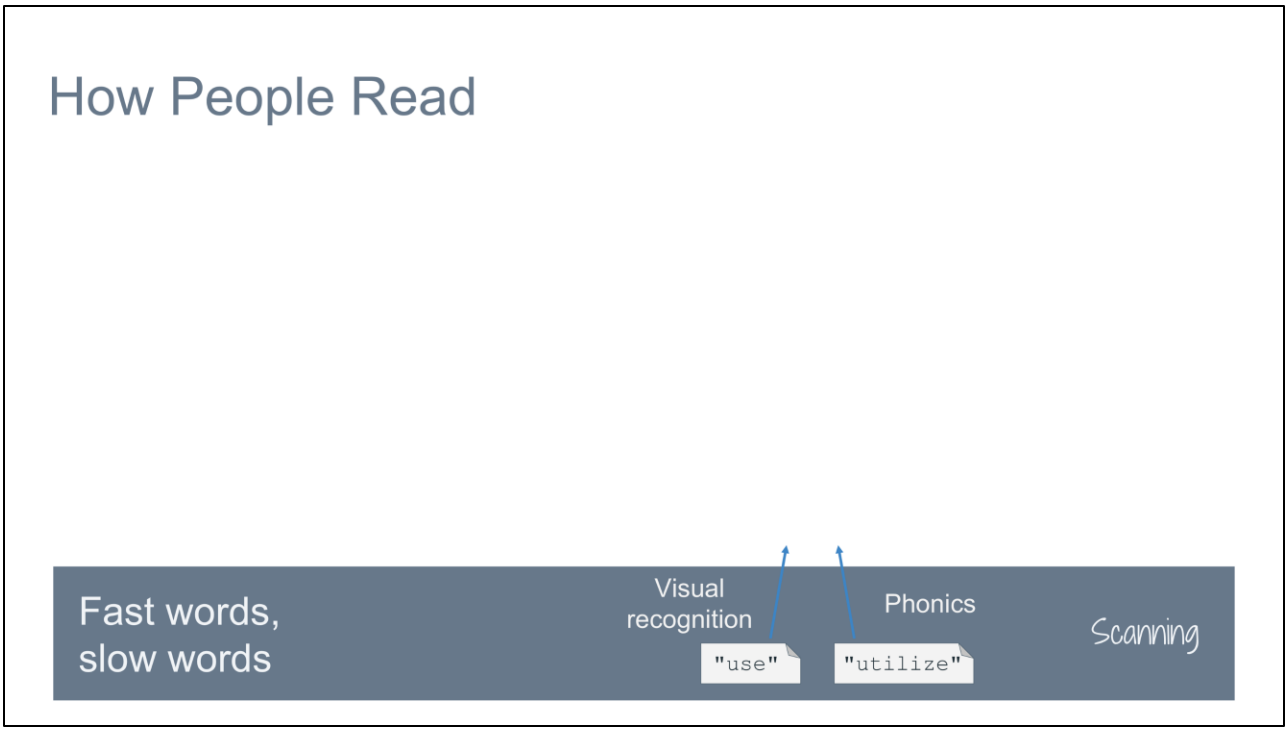
We'll mostly talk about writing and editing wiki docs, but the ideas also apply for many other formats.

The talk takes about half an hour. Most sections take about five minutes, but we'll spend ten minutes on Structure.

We have a lot to cover, but it all fits together, and at the end we'll do a quick review.



In this first section, we'll talk about what makes writing "good". We'll start by understanding how people read.

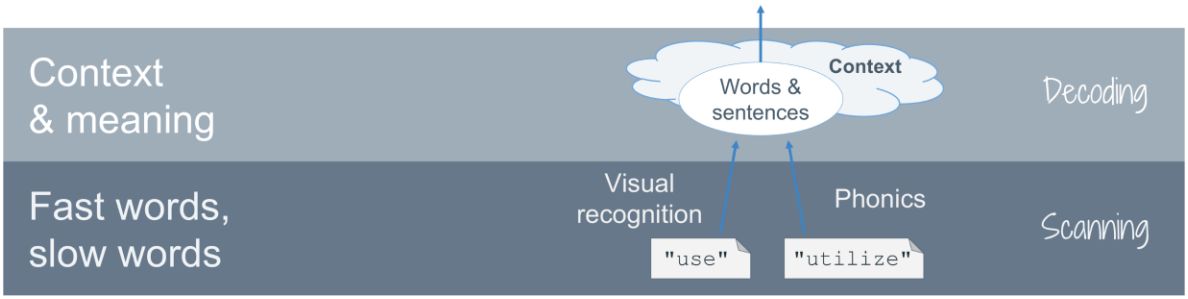


The brain has two strategies for reading a word. With a short, familiar word like *use*, our neural networks instantly recognize the word from years of practice. But with a longer or less-familiar word like *utilize*, we may have to *decode* it using the rules of English phonics.

That takes more work - especially for the high percentage of engineers who are "ESL", meaning that they speak English as a Second Language. It's also more work for the 20% of engineers with some level of *dyslexia*, a trait that makes phonetic reading difficult. But it's not just ESLs and dyslexics. The visual recognition path is faster for *everyone*. So in this talk we'll suggest using short, familiar words when you can.

~~~

<NOTE: The visual vs. phonetic paths correlate to the battle between two methods of teaching children to read, e.g. in At a Loss for Words: How a Flawed Idea Is Teaching Millions of Kids to Be Poor Readers (American Public Media 2019). For more on the high incidence of dyslexia among engineers, architects, artists, and entrepreneurs, which may be twice the normal percentage, see: The Dyslexic Advantage by Brock & Fernette Eide (p. 4 and Appendix B), the paper "Dyslexia in high performers - a study across 4 degree disciplines", the article "Contemplating the correlations between engineering and dyslexia", and Proust & the Squid: The Story & Science of the Reading Brain chapter 8 "Genes, Gifts, and Dyslexia".>
~~~

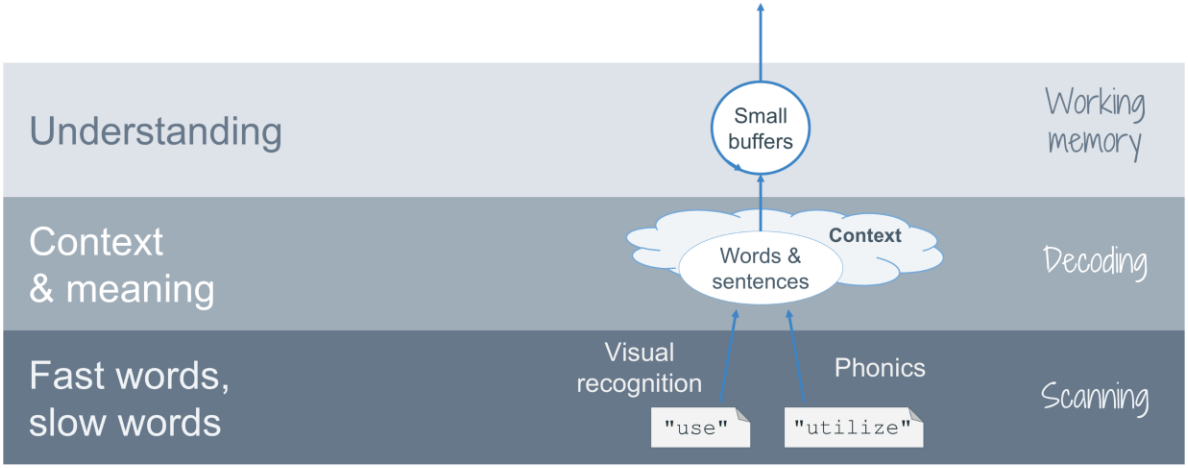
How People Read



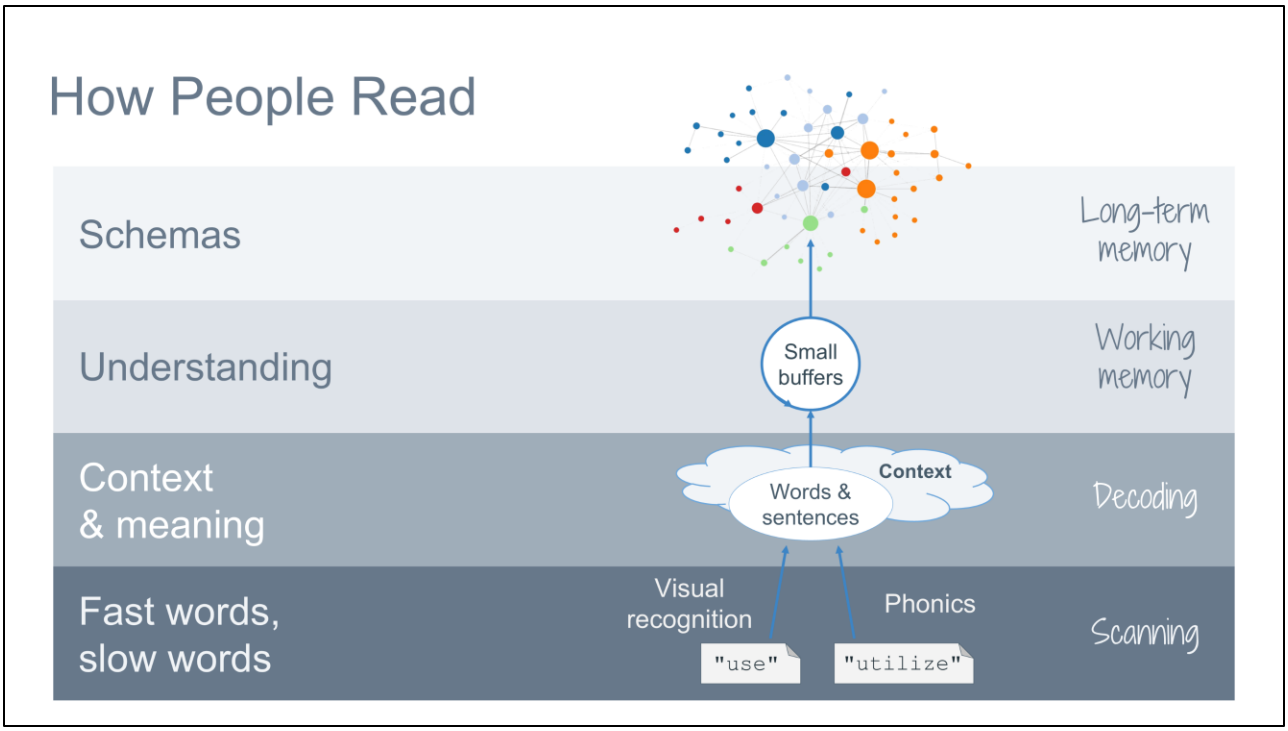
Reading is massively parallel, involving many parts of the brain that all influence each other. As a result of that mutual influence, *context* has a huge impact on how we understand what we read. So in this talk we'll encourage you to provide context, with things like *topic headings* and *introductions*.

~~~

<NOTE: For more on the power of context and reading goals in determining what we see, see the chapter on Reading Comprehension in The Reading Mind, e.g., p. 121 & 125.>
~~~

How People Read



The words you mentally hear as you read are picked up by your *working memory*. Working memory has very small *buffers* for audio and visual input. How well can you remember a sentence you just heard? How many pennies can you count in one quick look? That's about the size of the buffers. So we'll encourage you to use short sentences and small groupings to use those little buffers effectively.



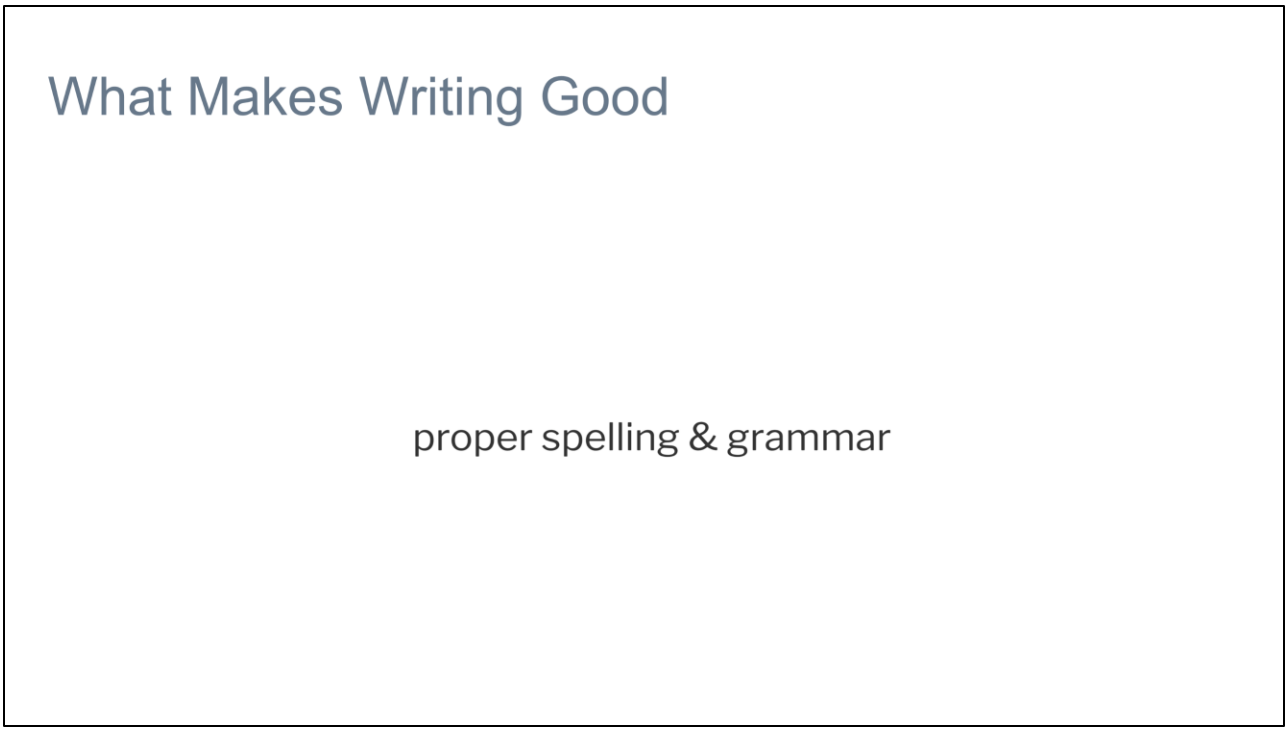
Knowledge is stored in long-term memory as *schemas*, similar to the schemas or graphs we use in computing. Readers don't remember every sentence on the page, just a *schema* of the main points. So we'll explain how your doc can help the working memory create schema - for instance, by providing a logical structure with well-defined units and clear connections.

That's the human reading stack.

Now, what makes writing "good"? Think back to a document that you liked. Try to remember why you liked it...

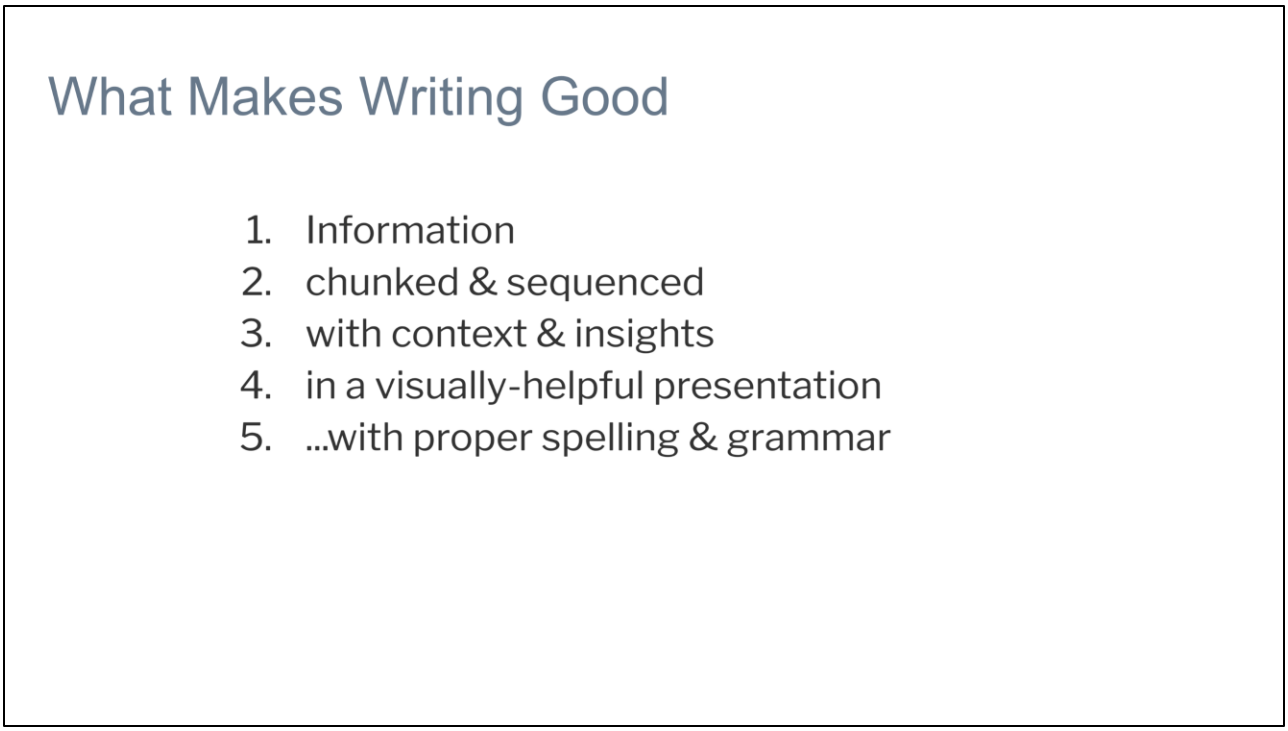
~~~

<NOTE: For a 15-minute overview of the science of reading, see A Cognitive Model of Readability. You'll find a detailed breakdown of working memory and schema under "Cognitive Load Learning Theory" on p. 104-112 of Content and Complexity (2003). For schema theory see p. 32-33 of the free textbook How People Learn by John D. Bransford et al. Schema theory and the Curse of Knowledge are core concepts in the book Made to Stick: Why Some Ideas Take Hold and Others Come Unstuck.

All these theories are "cognitive" (high level, not neurological). We now know from PET scans that the working memory is managed by an "executive" in the prefrontal cortex, and the "buffers" are the actual sensory short-term memory areas elsewhere
~~~

in the surface of the brain (e.g., visual cortex, auditory cortex, etc.). This means that the executive is wired as an input source for short-term sensory memory, much like the eyes, ears, skin, muscles, etc. Schema theory is 100 years old; working memory theory about 40. For other sources search for schema theory, working memory, or cognitive load.>



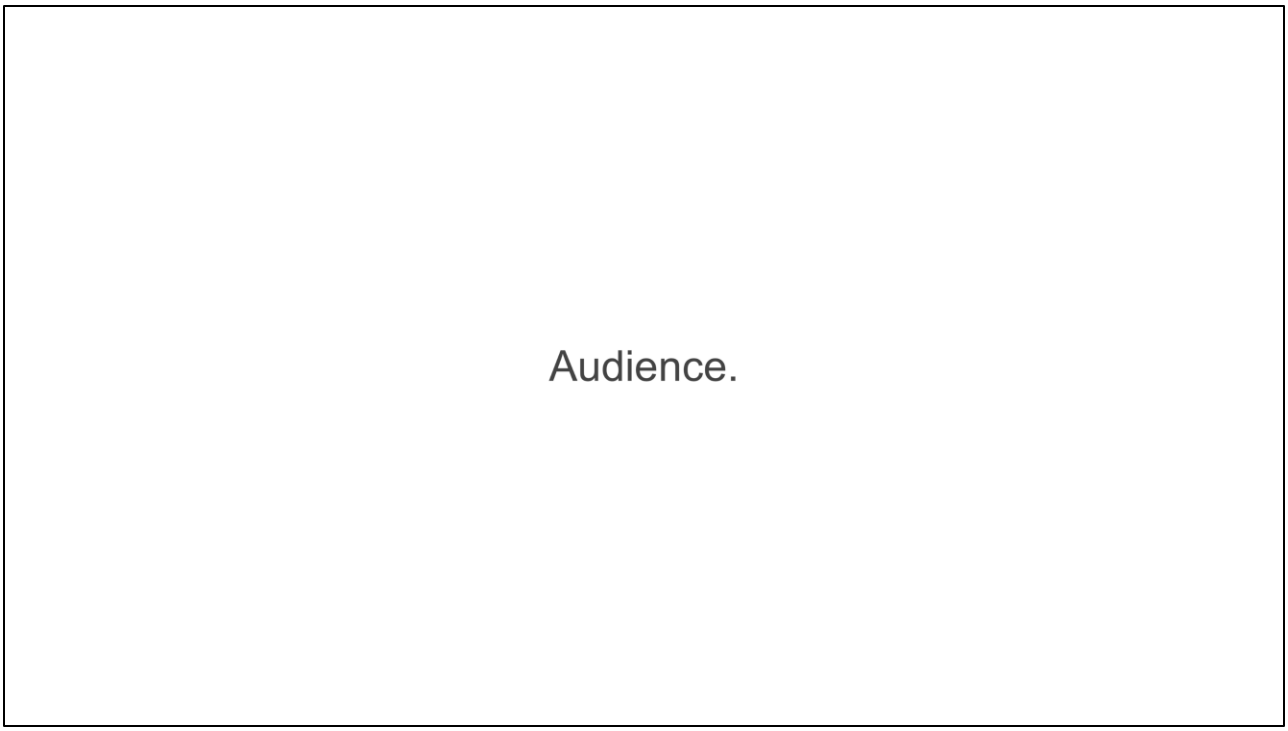
It probably wasn't this! "Proper spelling and grammar" are nice, but is that really what makes writing *good*?

Here are some things that *tech writers* think make writing "good".



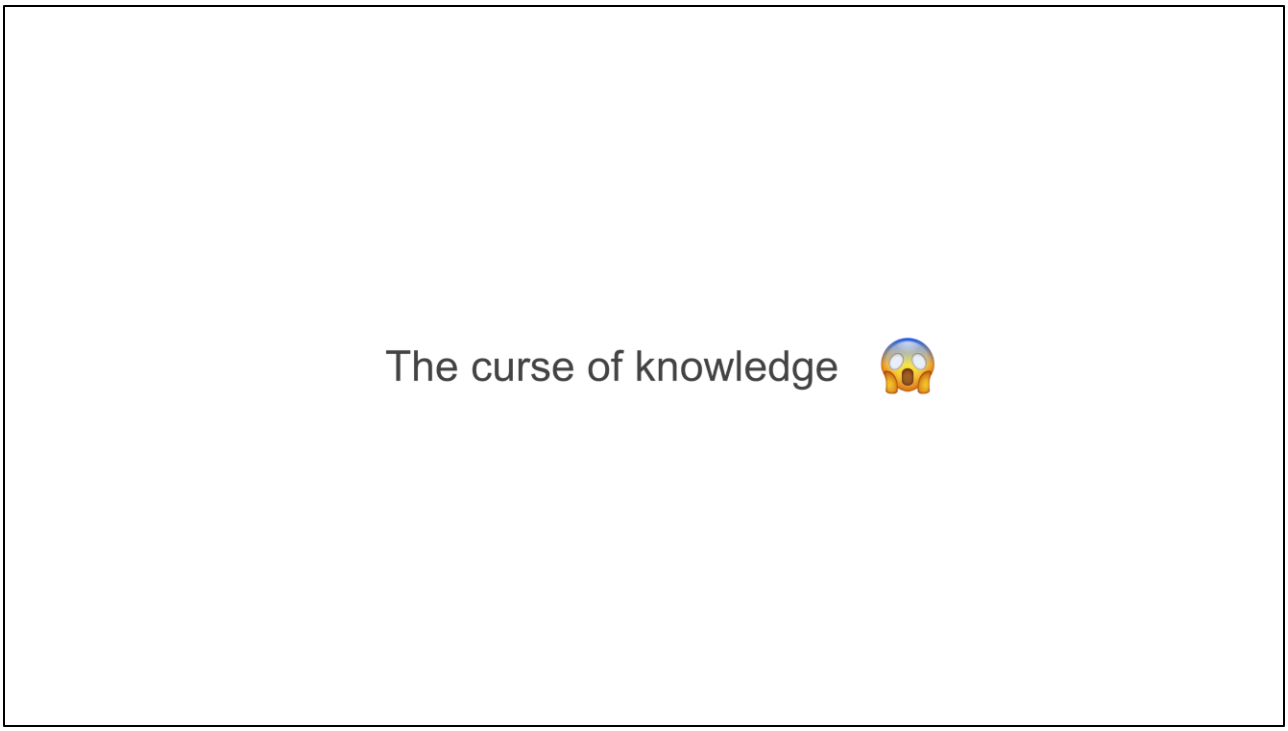
- We need reliable, accurate information.
- And not just one giant paragraph in random order. It needs to be *structured*, or logically grouped and sequenced.
- And please, not just a dry list of facts! We want tips and insight from your experience.
- Also, we like a little help from features like tables, images, code samples, and so on.
- Spelling and grammar are at the bottom of the list, and are the easiest to fix if necessary.

Now, here's the number one thing that we writers mention to engineers:



Audience! Audience is huge. In any project, it's one of the first things a writer asks about. Understand your audience, and write for them.

When thinking about your audience, watch out for this:

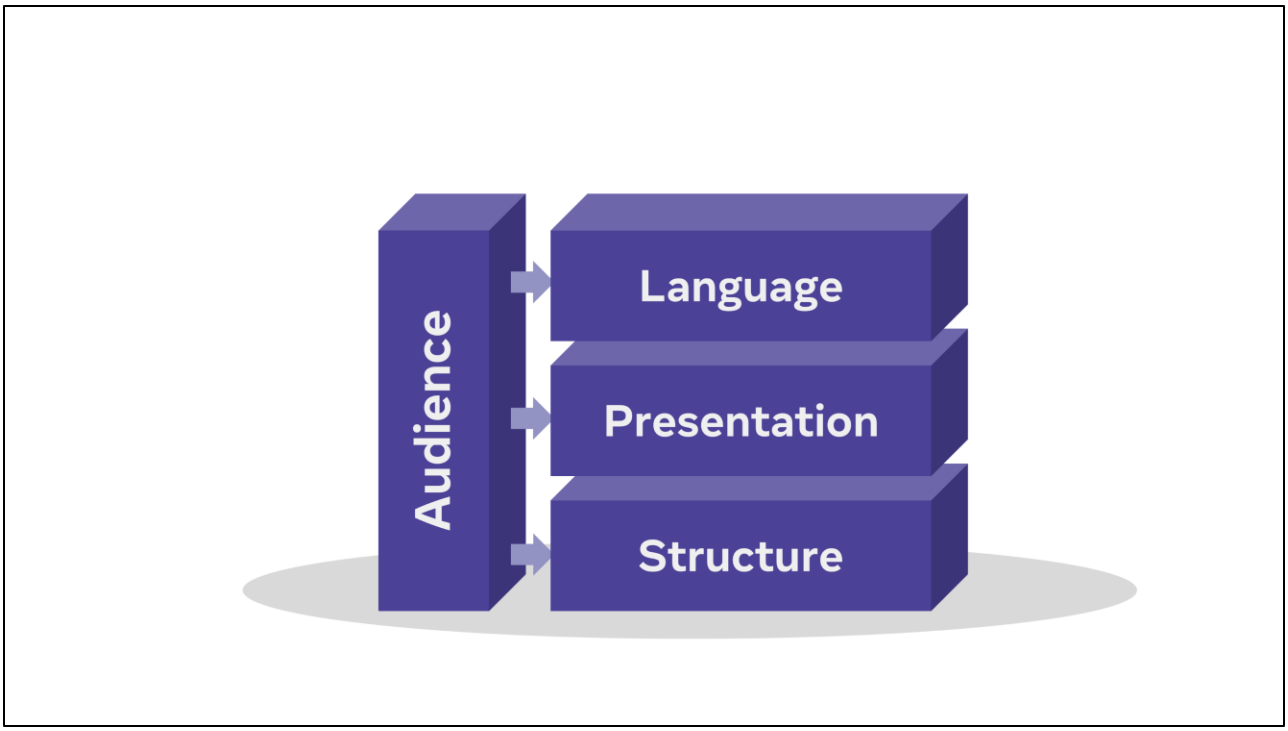


When you know a lot about something, there's an unconscious tendency to think other people do, too. Psychologists call this the *curse of knowledge*, and it's been demonstrated many times.

In one study, iPhone users were asked how long it would take new users to learn a specific iPhone feature. The most *experienced* users consistently *under*estimated the time and difficulty. But *newer* users estimated the learning time correctly.

The *curse of knowledge* is an unconscious mental bias that can make experts talk at a level their audience won't understand.

~~~
*<NOTE: There's a Wikipedia article on the curse of knowledge.>*
~~~



Audience impacts everything: the way we *structure* the docs, the way we *present* the information, and the *language* we use to communicate.

Think About Your Audience

So, who will read your doc… and why? What's their use-case? Some readers want a full explanation before they start a big project. Others are fixing a critical bug at 3am, and just need to find a quick answer.

What does your audience already know, and what is their *path* through your content?

Are your readers on *your* team, or *another* team? Other teams don't need your team's internal docs, such as oncall docs and new-hire docs, so we recommend keeping those team docs separate.

We suggest writing at a technical level that new engineers can understand. But we often try to be readable for PMs and designers, too.

Along with audience, think about your doc's intended *impact*. What is it, and will you be able to measure it to know if your doc is successful?

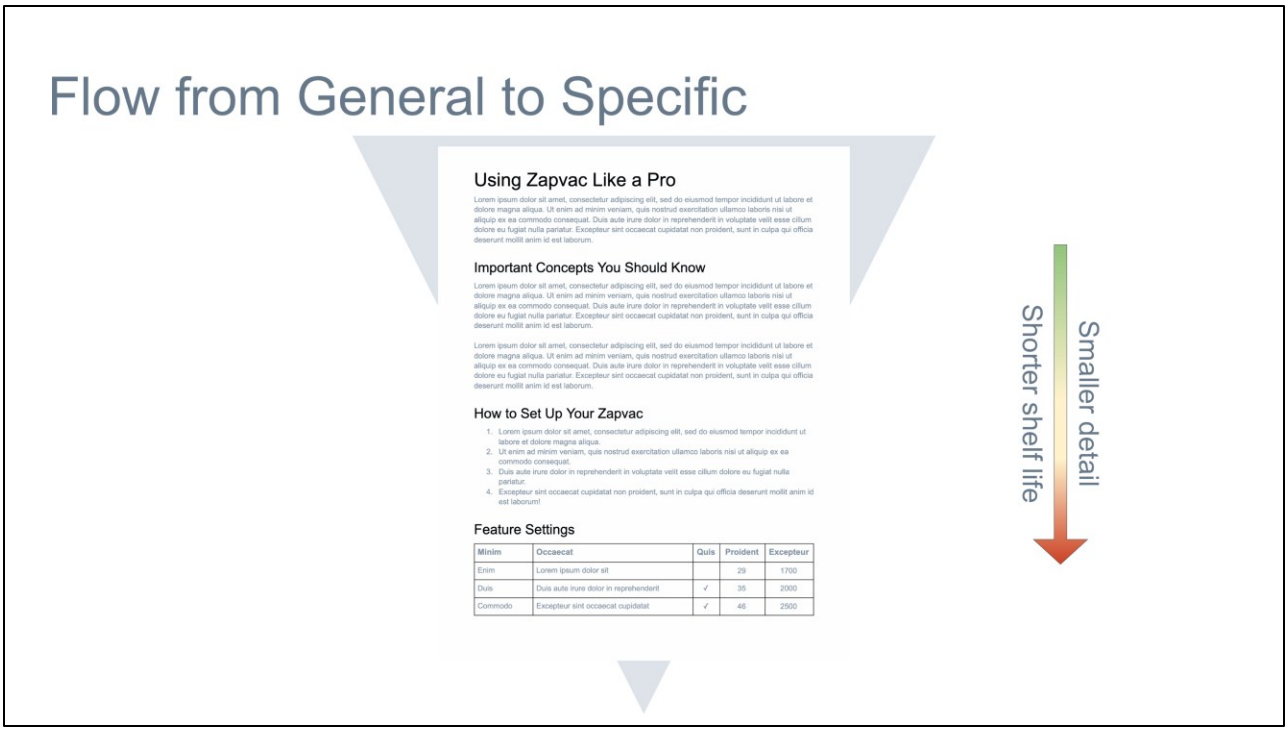
In some cases, you just need to capture knowledge before you forget it or move to a new project. That kind of *brain dump* has value for you and maybe your team, even if you don't take time to organize it logically.

But most audiences need a *structured reading experience*. In the next section, we'll show how to design one.

With a good *structure*, almost any doc problem can be fixed. But a *chaotic* structure makes the doc hard to write, hard to read, and hard to fix. So we'll spend the next ten minutes talking about structure.

We'll start with 3 principles for creating *modular* docs, then talk about designing a doc for *maintenance* and *discovery*, and show you a powerful trick for *outlining*.

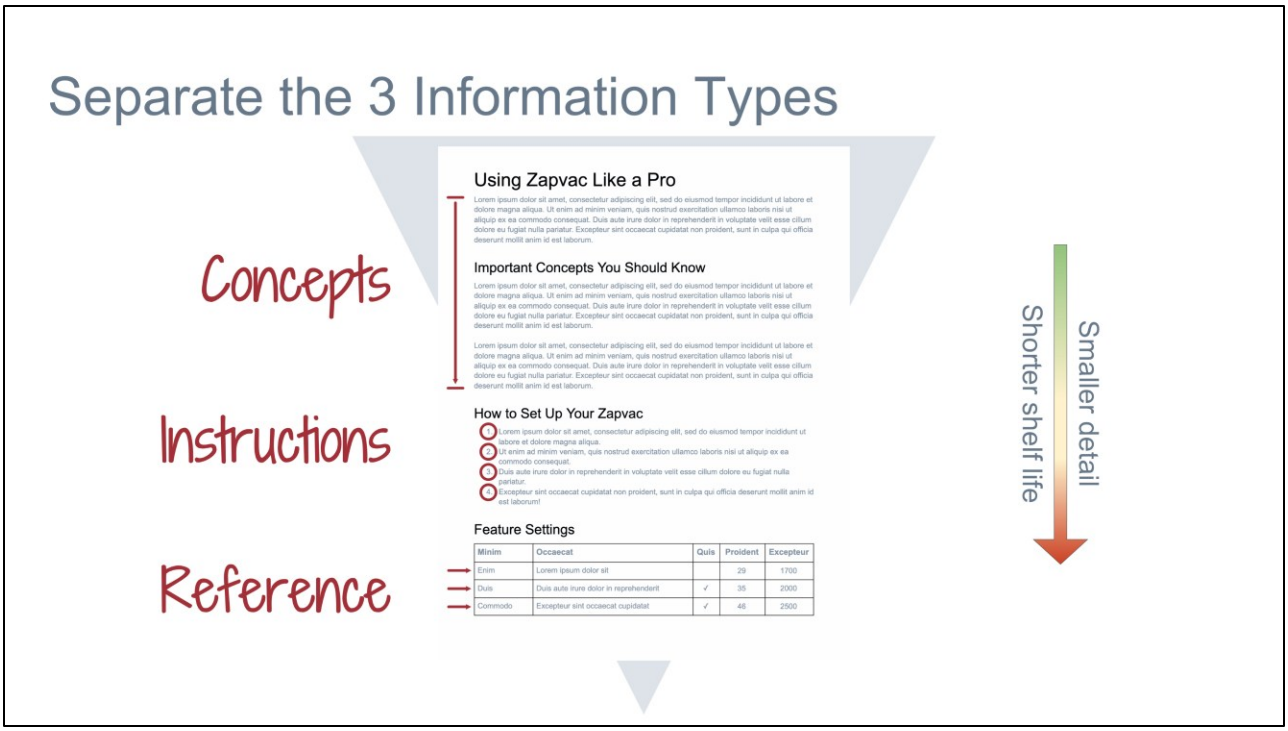
~~~
*<NOTE: Wikipedia has a good article on Information Design that introduces many related topics.>*
~~~



First, technical documents tend to flow from the most general information (at the top) to the most detailed (at the bottom). If you're explaining computers to someone, you don't start with circuit boards. You start with big ideas, like the definition of computing, and then work down to details. You're creating a mental framework, then populating it with information. In many cases, we use this same pattern in each *section* of the doc, too.

One side-benefit is that this serves multiple audiences. Some readers, such as a PM, may just want the big picture, which they get in the first few paragraphs. Others may read the whole doc, start to finish. And some may come back later to look up specific details at the bottom.

Notice that big concepts change slowly. Low-level details are closer to the code, and tend to change more frequently. For instance, the features of a word processor or spreadsheet app may change a lot over the years, but the basic concept remains pretty stable.



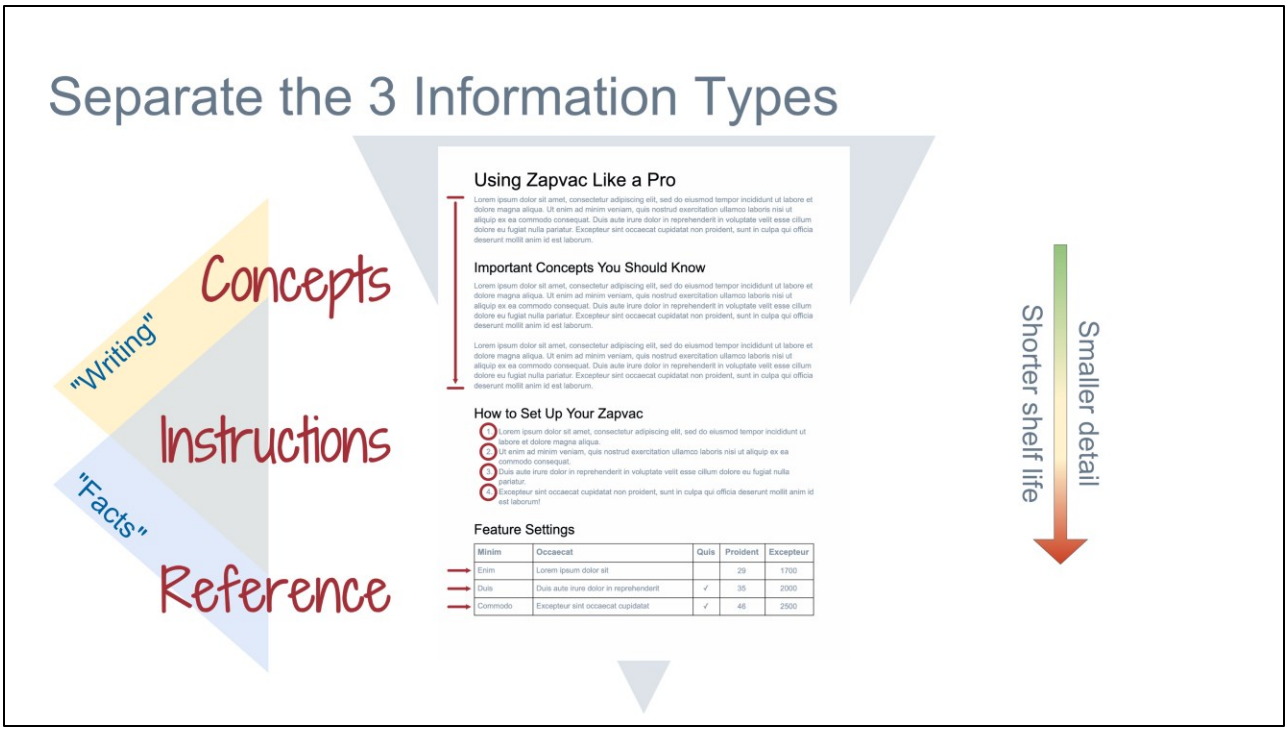
Second, try to separate the 3 information types.

- *Concepts* are written as paragraphs in logical sequence, and people often read them from start to finish.
- *Instructions,* like a tutorial or installation guide, are numbered to help readers keep track of progress as they work.
- *Reference material* is more like a dictionary - designed for random access, not sequential reading, and often listed in alphabetic or numeric order.

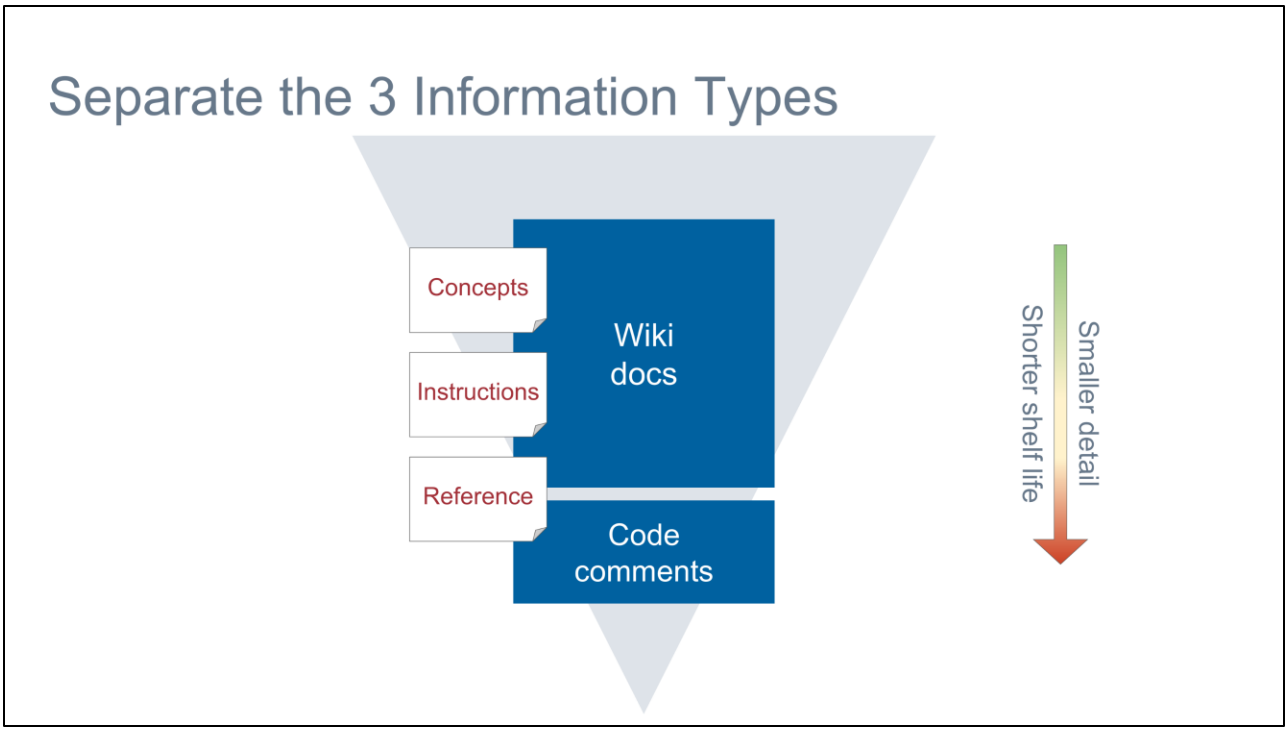
This model works well with our flow from general to specific. *Big ideas* are at the top, then *smaller concepts* or *instructions*, then detailed *reference* material.

It's hard to work with a doc where the 3 types are mixed together without clear boundaries. For example, when instructions are mixed with deep conceptual explanations, it's hard for a reader who wants just the concepts, or just the instructions. In most cases, it's best to keep them separate, like you see here.

~~~

<NOTE: The concept/task/reference model is taken from the DITA XML schema for tech writing. For simplicity, these slides use the word "instructions" instead of the DITA term "tasks".>
~~~



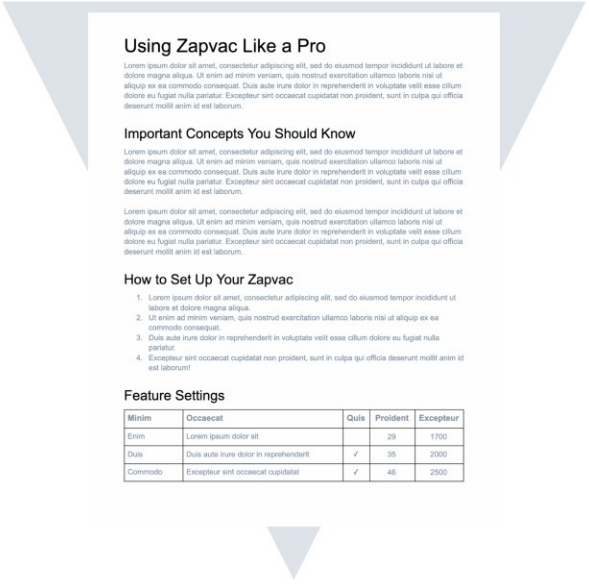
The *reference* material is often just simple facts, so it may be the easiest part to write. A *concept* may require more difficult writing, even when it looks easy. For this reason, some people write the *introduction* last.



Sometimes detailed *reference* material is stored in the *code* as comments. Engineers often do this when building APIs, for instance. Those code comments enable them to auto-generate attractive, organized documents with tools like Javadoc or Sphinx.

Avoid Repetition

"x = 7"



Using Zapvac Like a Pro

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Important Concepts You Should Know

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

How to Set Up Your Zapvac

1. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
2. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
3. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.
4. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum!

Feature Settings

Minim	Occaecat	Quis	Proident	Excepteur
Enim	Lorem ipsum dolor sit		29	1700
Duis	Duis aute irure dolor in reprehenderit	✓	35	2000
Commodo	Excepteur sint occaecat cupidatat	✓	46	2500

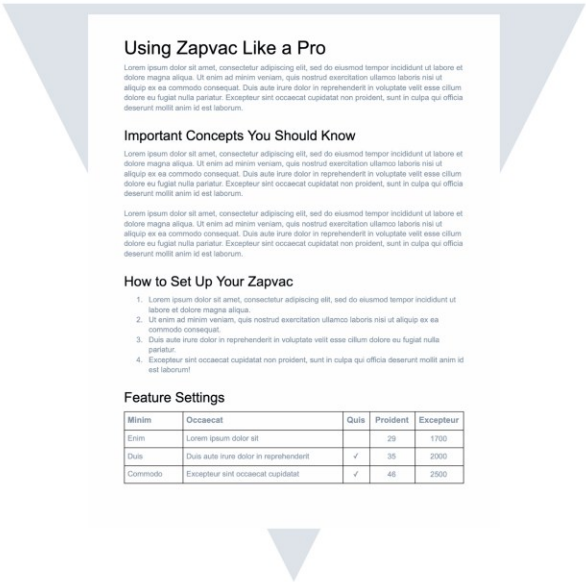
Our *third* principle for modularity is to avoid repetition. Try to define or explain a thing in *just one place*. Do this with concepts, instructions, parameters, and so on. People only have to read it once, you can update it in just one place, and you'll never have two conflicting versions.

Avoid Repetition

"X is the…"

"Set x & y."

"x = 7"



Smaller detail

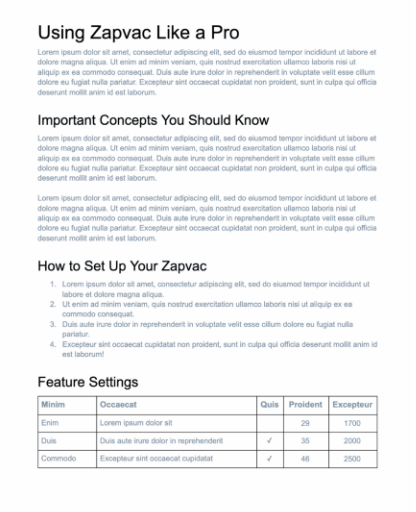
Shorter shelf life

Other parts of the doc may *reference* something without repeating it. Duplicate information is inelegant, like duplicate functions or data. It's hard to maintain, and it creates redundant search results.

We also suggest that you skip *unnecessary* information. Studies show that removing *non*-essential information helps readers learn the *essential* information better. If it's not important, move it to an appendix, or delete it so no one has to maintain it.

Modular Docs

1. Flow from general to specific
2. Separate the 3 information types:
   - Concepts
   - Instructions
   - Reference
3. Avoid repetition



Using Zapvac Like a Pro

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Important Concepts You Should Know

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

How to Set Up Your Zapvac

1. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
2. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
3. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.
4. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum!

Feature Settings

Minim	Occaecat	Quis	Proident	Excepteur
Enim	Lorem ipsum dolor sit		29	1700
Duis	Duis aute irure dolor in reprehenderit	✓	35	2000
Commodo	Excepteur sint occaecat cupidatat	✓	46	2500

Okay. The 3 principles we just discussed (general-to-specific flow... the 3 information types... and avoiding repetition) lead to a *modular* document. Modular docs are like well-structured code: easy to navigate, read, write, and maintain.

What should we talk about next?


How about doc maintenance?
Or how readers will find a fix for a critical bug at 3am?

Hmm.

Plan for Maintenance

- ☑ Use modular design
- ☑ Plan for updates & growth
  - ☐ Text & tables
  - ☐ Images
  - ☐ Sample code
  - ☐ Page style

As you work on your doc, think about *maintenance*.

Your *modular design* means you can often update just one module, without touching the rest of the doc. Visitors will even make those updates for you. (We love it when people to do that!) And it's easy to add new features and sections. A good doc design is *extensible* and *open-ended*.

Images like diagrams and screenshots add a lot, but they have to be updated to stay current. So use a calendar, task scheduler, or other tool to remind you to check them at least every six months. Store the diagram in a shared folder so your teammates can update it. And use a popular drawing app that everyone can use, like Google Draw, Powerpoint, or Excalidraw.

In surveys, engineers tell us they love *code samples*. And *tables or charts* can make your documents clear and convincing. So definitely use code samples and data, but remember to keep them updated! Use a calendar or other tool to remind you, or attach that task to other procedures so it’s automatic.

Plan for Maintenance

- ☑ Use modular design
- ☑ Plan for updates & growth
  - ☑ Text & tables
  - ☑ Images
  - ☑ Sample code
  - ☑ Page style

Finally, go easy on your *page styling*. Wikis and internal documentation are often about speed and community editing. Don't build an art project that's hard for others to maintain. Maybe use custom style for *landing pages*, but not for *content* pages. Research shows that consistent visual style improves reading speed and memory.

Modular design, simple styling, and regular updates to images, samples, and data… You don't have to remember all that.

Just remember, as you work, to think about how someone will maintain your doc.

~~~

<NOTE: For the wiki philosophy, see Wiki in Wikipedia.>
~~~

Plan for Discovery

- ☑ Invest in navigation
- ☑ Optimize for search
- ☑ Show reading paths
- ☑ FAQ

Next, help people *find* your content. You might first try to understand how much people find things with *search* versus *navigation* (that is, menus, links, etc.).

You might be able to do this by analyzing traffic logs. Do page hits come from outside the site or inside the site? From search pages or search boxes in the site, or from navigation links? Or perhaps from AI recommendations? You might want to add URL parameters to the hyperlinks in your site to make it easy to track this.

One common model is for people to arrive in a general area by search, and then use navigation to close in on their target.

So optimize your page for Search. Give it a readable, searchable title, and make sure your page uses the words that people will search on. A good title usually has more than one word, but not too many to in the search results or navigation display.

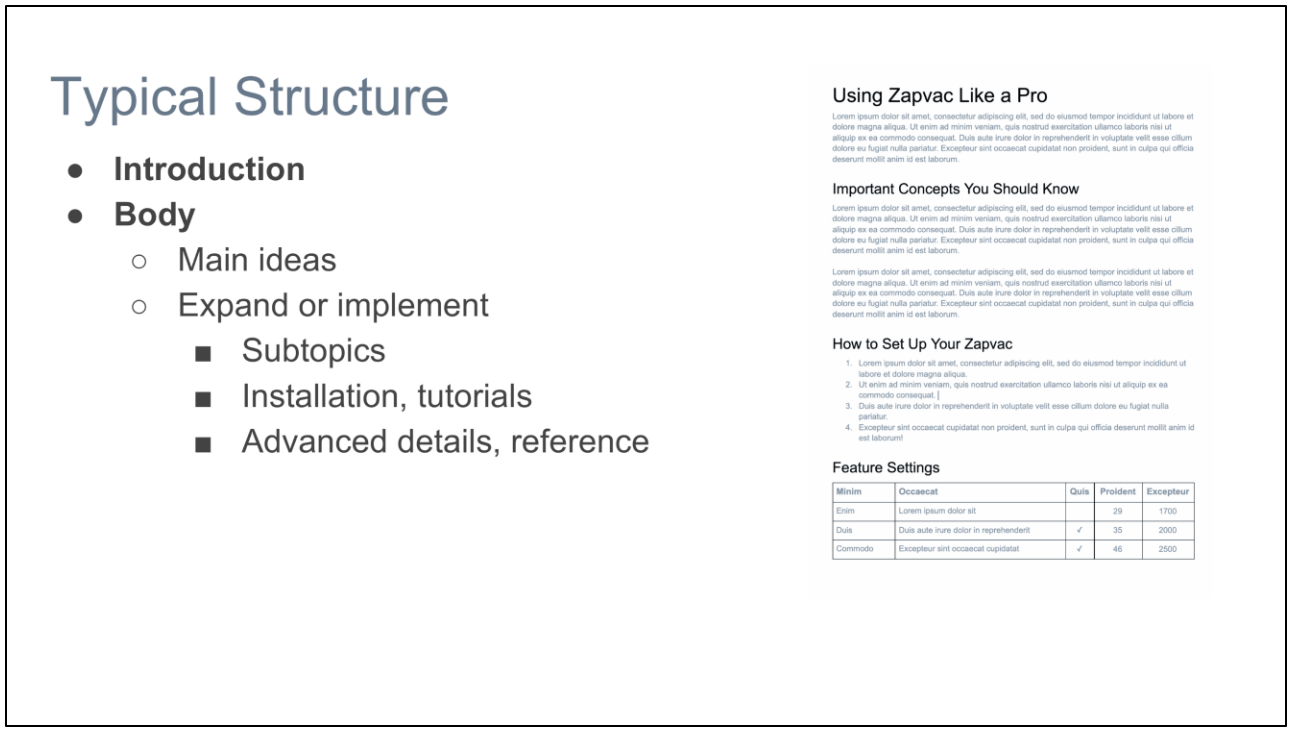
Use common terms on your page, not just your acronyms and product names. People who haven't read your page may not know your acronyms and product names, so they can't use them to search! :)

Your clear page titles will look good in your navigation, but you must also give your navigation an intuitive, logical structure. Your wiki or CMS may lst you use tags to generate navigation. And give the page itself a clea structure with clear, simple section headings. (I'm not a fan of clever or cryptic page titles or headings. They're

hard to understand and make your material hard to remember.)

Also make sure readers know which pages they should read, and in what sequence. You might have reading lists or special landing pages for specific audiences, use-cases, or topics. Or even just a list of related links on your content pages. Keep all these lists up to date!

Finally, consider using an FAQ. I'm not a fan of FAQs; to me they're an admission that your site doesn't have a logical place for everything. But sometimes they're a fact of life. In those cases, the FAQ is not necessarily part of a planned reading path. Instead, an FAQ item may come up when searching with a specific question. FAQs may also be a quick way to capture new learning and get it to people who need it. But in both of those situations, don't let the FAQ become a junkyard of duplicate or obsolete information. Stay on top of it, and keep duplicate info updated in the multiple places where it appears!

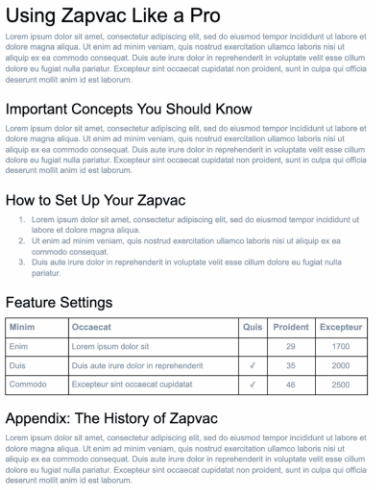


Here's a typical wiki page structure. It has an *introduction* followed by a *body* with several *sections*. It follows our general-to-specific pattern, and it separates the 3 information types.

This pattern works for a single wiki page, or an entire document set. In a wiki, readers can search a page and all its subpages from a single search box at the top of the page. So you can put everything on one page, or put each section on a separate page. People may find shorter pages less overwhelming, and be more likely to update them. On the other hand, having a lot of pages can make your structure and navigation complex. You’re the author; do what works best for your audience.

Typical Structure +

- **Introduction**
- **Body**
  - Main ideas
  - Expand or implement
    - Subtopics
    - Installation, tutorials
    - Advanced details, reference
- **Ending** *(optional)*
  - Summary, What's Next
  - POCs, Related Resources
- **Appendix** *(optional)*
  - Side concepts, reference, troubleshooting, FAQ



Using Zapvac Like a Pro

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Important Concepts You Should Know

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

How to Set Up Your Zapvac

1. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
2. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
3. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

Feature Settings

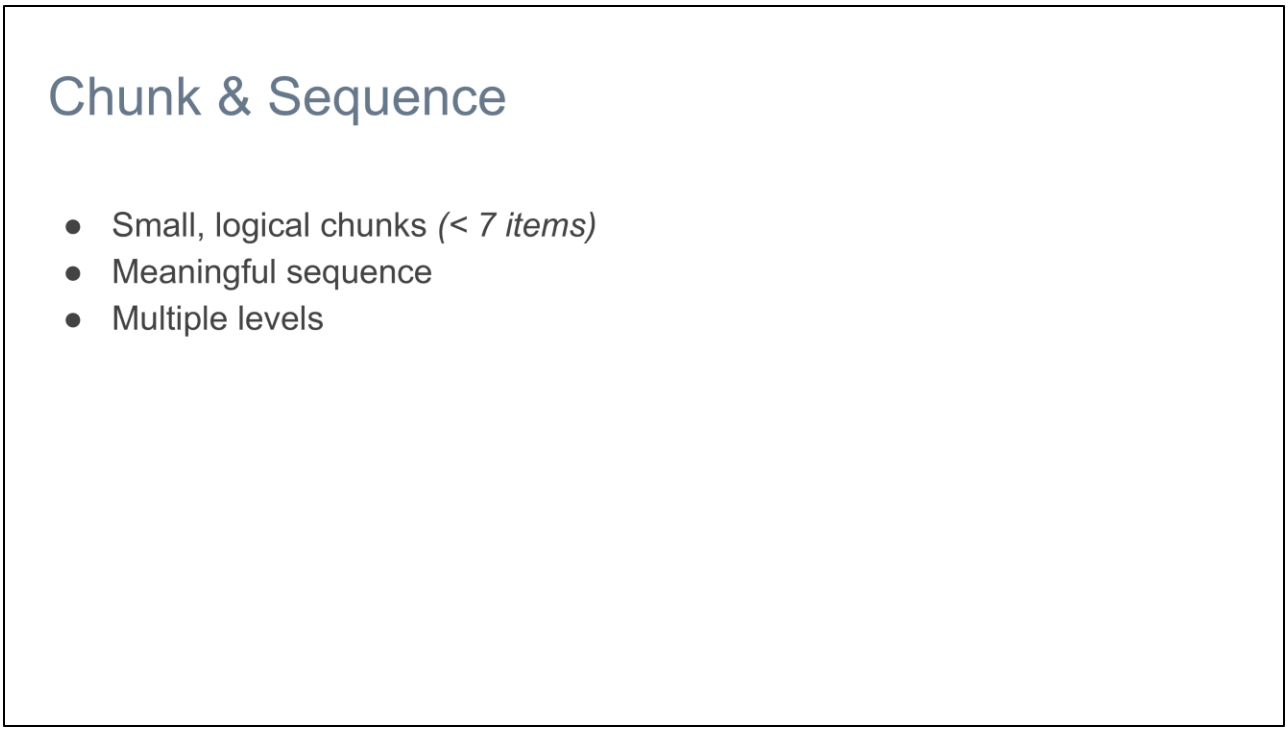
Minim	Occaecat	Quis	Proident	Excepteur
Enim	Lorem ipsum dolor sit		29	1700
Duis	Duis aute irure dolor in reprehenderit	✓	35	2000
Commodo	Excepteur sint occaecat cupidatat	✓	46	2500

Appendix: The History of Zapvac

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Here's the same template, with a little more at the bottom.

For instance, a long page might have a *summary* to tie things together. You might have a list of related links, such as a reading list, contacts, or groups. And an *Appendix*, or multiple *Appendixes*, can contain things that would break up your main flow. That might include long instructions, reference tables, background, etc. For those items, use a heading that starts with the word "Appendix:" to make it obvious.

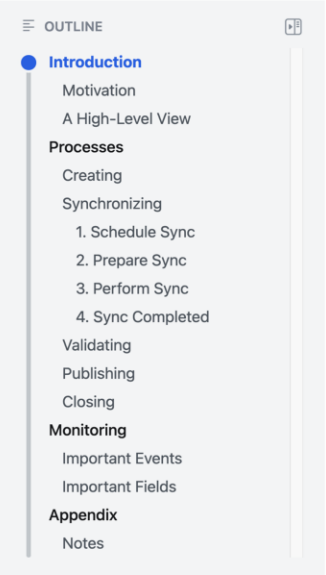


Do you remember *working memory*, which turns sensory input into *schemas* for long-term storage? It has small buffers that hold a few seconds of audio, a simple image, or a short list of 4 to 7 items. Anything more than that takes a conscious effort. So it's constantly compiling input into units called *chunks*, and looking for meaningful ways to *connect* them.

Studies show that people can easily memorize very long lists, if the list is organized as *small groups* or *chunks* in a *logical sequence*. That's how the mind works. So as writers we *chunk* our docs into sections, paragraphs, and so on, and we create a logical *sequence* for everything. We're *pre-schematizing* the page for the reader.

Chunk & Sequence

- Small, logical chunks *(< 7 items)*
- Meaningful sequence
- Multiple levels



It's hard to read a flat table of contents with 20 headings at the same level. But it's easy to read one with small subgroups in a logical sequence. The reader can see the big picture. This helps them patiently read the entire doc, or quickly find the thing they need.

If you have a long list of bullets or numbered instructions, break it into meaningful sections. If you have a complicated diagram, group related items together visually. You can chunk and sequence almost anything. The main exception is a long lookup table or dictionary, which can be flat.

A Structure Algorithm

1. List the topics
2. Chunk & sequence
3. Solve overlaps & dependencies

You already know how to outline. Just list the topics you need to cover, then chunk and sequence the list. Make sure each sub-list is complete. For most docs, the structure is pretty obvious. Other docs will make you work for it… but those tough cases are the ones when a good outline will 10x your doc.

A Structure Algorithm

1. List the topics
2. Chunk & sequence
3. Solve overlaps & dependencies

Fix path problems by using levels.

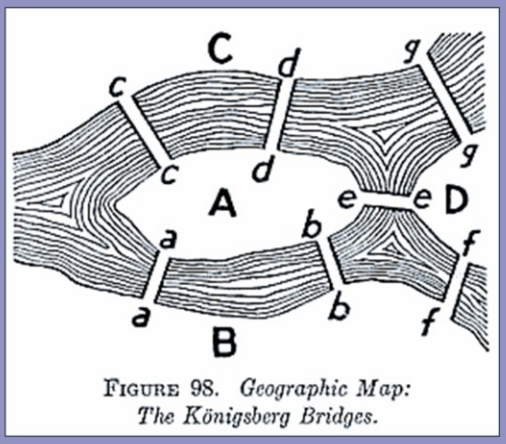


FIGURE 98. *Geographic Map: The Königsberg Bridges.*

Two problems come up…

Sometimes a topic could go in more than one section. Since you want to avoid repetition, try to choose just one.

A harder problem is to resolve complex inter-dependencies. If two topics depend on each other, which one do you explain first? It's a graph theory problem, like the Seven Bridges of Konigsberg. Is there one path that connects all your points without repeating any of them? Sometimes there's no perfect solution. In those cases, approach it at multiple levels, Use an overview and sometimes "basic" and "advanced" sections to gradually fill in the details. We'll talk about this again.

~~~

<NOTE: The Internet has many good resources on taxonomy and ontology (two concepts similar to outlines) and how they relate to documentation and knowledge management. For our purposes, a "taxonomy" or "ontology" covers all the information and provides a place for everything, but is not sequential like a reader path, and may have more than one logical place for some things. For instance, in this talk the maintenance of diagrams could be mentioned in the Maintenance section or under Diagrams. The author has to decide and try not to be too repetitive.
~~~

A Writing Algorithm

1. Structure
2. Draft
3. Review

Try to separate these processes.

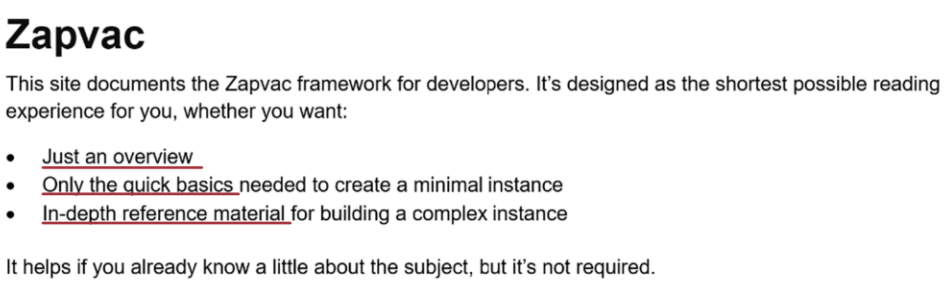
When your structure is done, write a draft by filling it in with content.

Here's a pro tip: A lot of writers suggest that you finish your *structure* before you write any *text*. And then finish your *first draft* of the text before you go back and *edit* it. This can save you from getting trapped in details, or letting your *critical* mind shut down your *creative* mind. But do what works for you!

~~~

*<NOTE: For thoughts on being blocked by the critical mind and the value of getting out even a lousy first draft, see Anne Lamott's popular short book on Bird by Bird, e.g. Chapter 3, "Sh*tty First Drafts". (Be warned that this book is more about writing fiction than non-fiction.)>*
~~~

Example: 20-Page Wiki

Challenge: Multiple audiences at multiple depths



Zapvac

This site documents the Zapvac framework for developers. It's designed as the shortest possible reading experience for you, whether you want:

- Just an overview
- Only the quick basics needed to create a minimal instance
- In-depth reference material for building a complex instance

It helps if you already know a little about the subject, but it's not required.

To finish this section on structure, we'll look at an example.

This sample describes a software framework used by some software engineers in a big company. The two main audiences are the engineers who *support* the framework and the engineers who *use* it. Most readers just want the basic concepts, a quick-start tutorial, and some sample code. But a few need to know everything. So for the engineer who wrote this wiki, the challenge was to serve several audiences at several levels.

Example: 20-Page Wiki

- Concepts
- Tutorial
- Key Topics
- Reference

Here's the high-level structure she came up with.

You can see that it starts with concepts, flows from general to specific, and keeps different types of information separate.

After the Concepts section and optional Tutorial, the Key Topics section can drill down into more detail. If she has to add more topics, she can add them there.

A Reference section at the end completes the set.

Example: 20-Page Wiki

- Concepts
- Tutorial
- Key Topics
- Reference

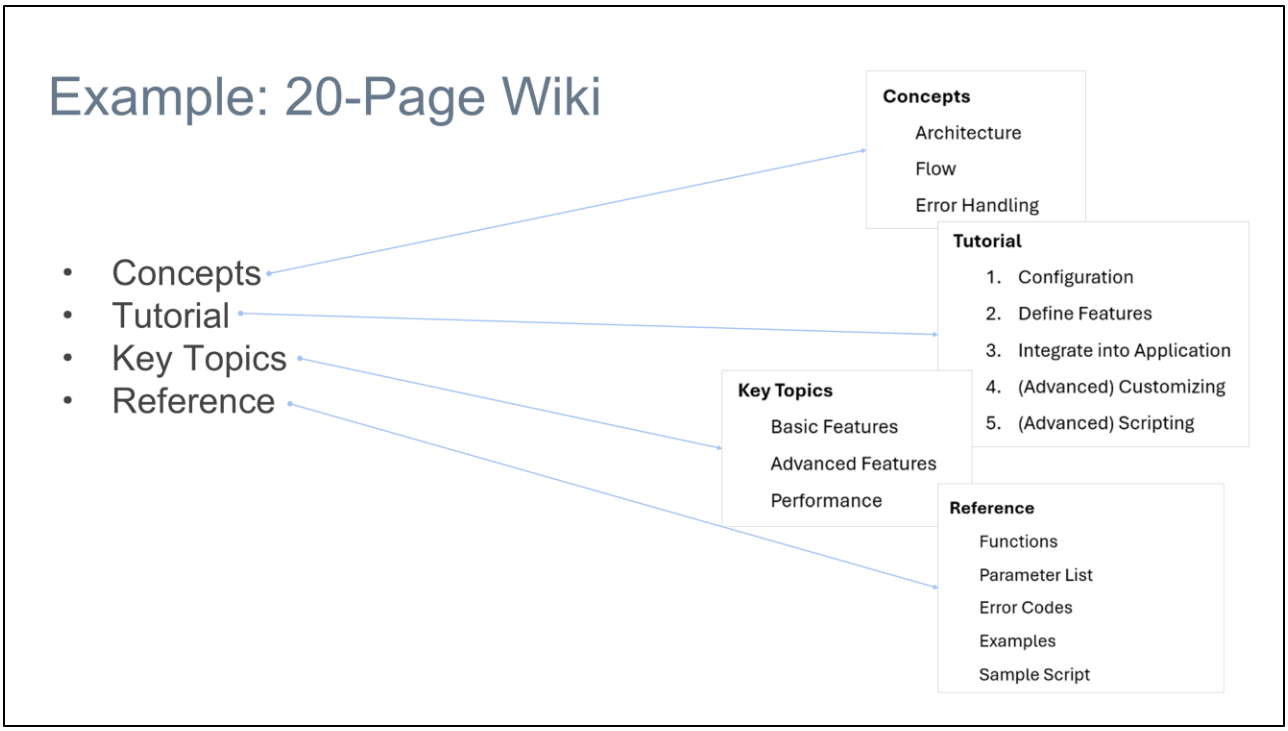
Concepts
- Architecture
- Flow
- Error Handling

Tutorial
1. Configuration
2. Define Features
3. Integrate into Application
4. (Advanced) Customizing
5. (Advanced) Scripting

Key Topics
- Basic Features
- Advanced Features
- Performance

Reference
- Functions
- Parameter List
- Error Codes
- Examples
- Sample Script

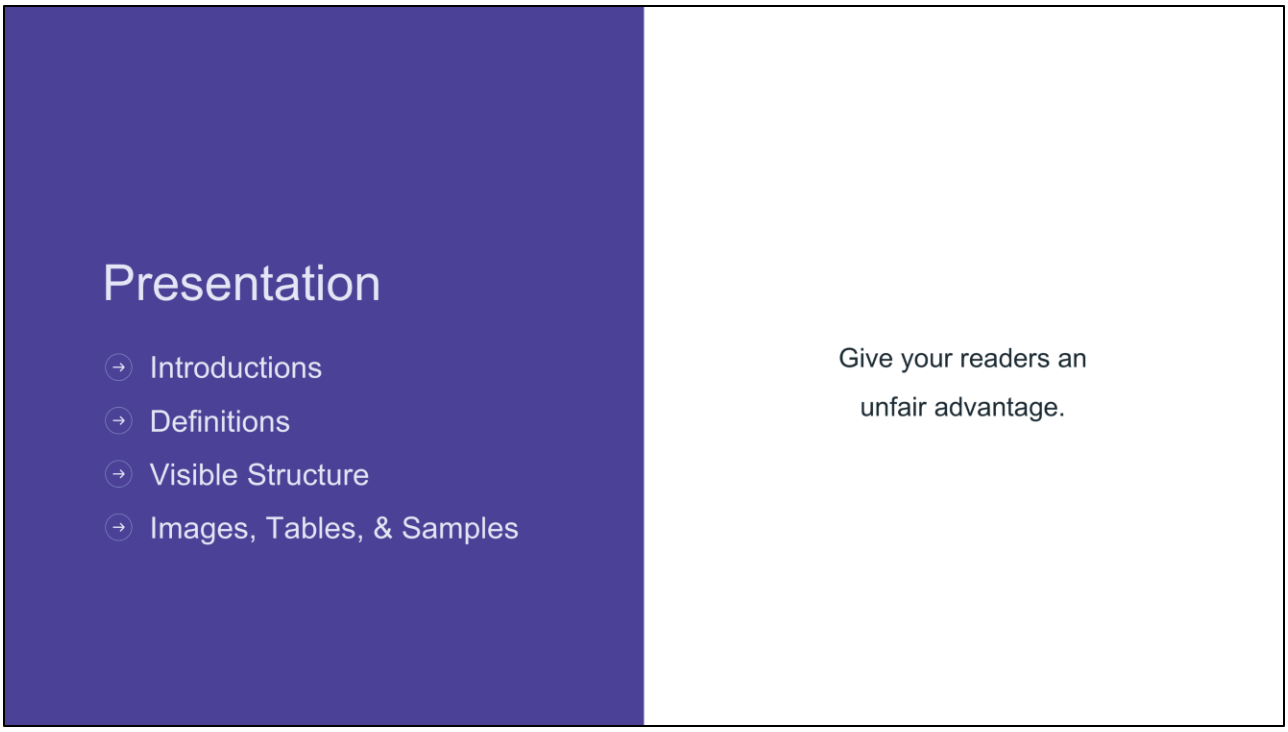
This is the full document structure, as seen in the navigation menu.



Each section lets people read just what they want. For instance, the *Tutorial* has meaningful numbered sections with clear titles, and the last two optional steps are marked for advanced study. The *Key Topics* section has separate headings for *Basic* and *Advanced* navigation. The *Detailed Reference* section has things that are less like big concepts, and more like lists or examples.

Designing a structure like this takes some work. But without this structure, writing and maintenance would be very tough, and the wiki would be much harder to read.

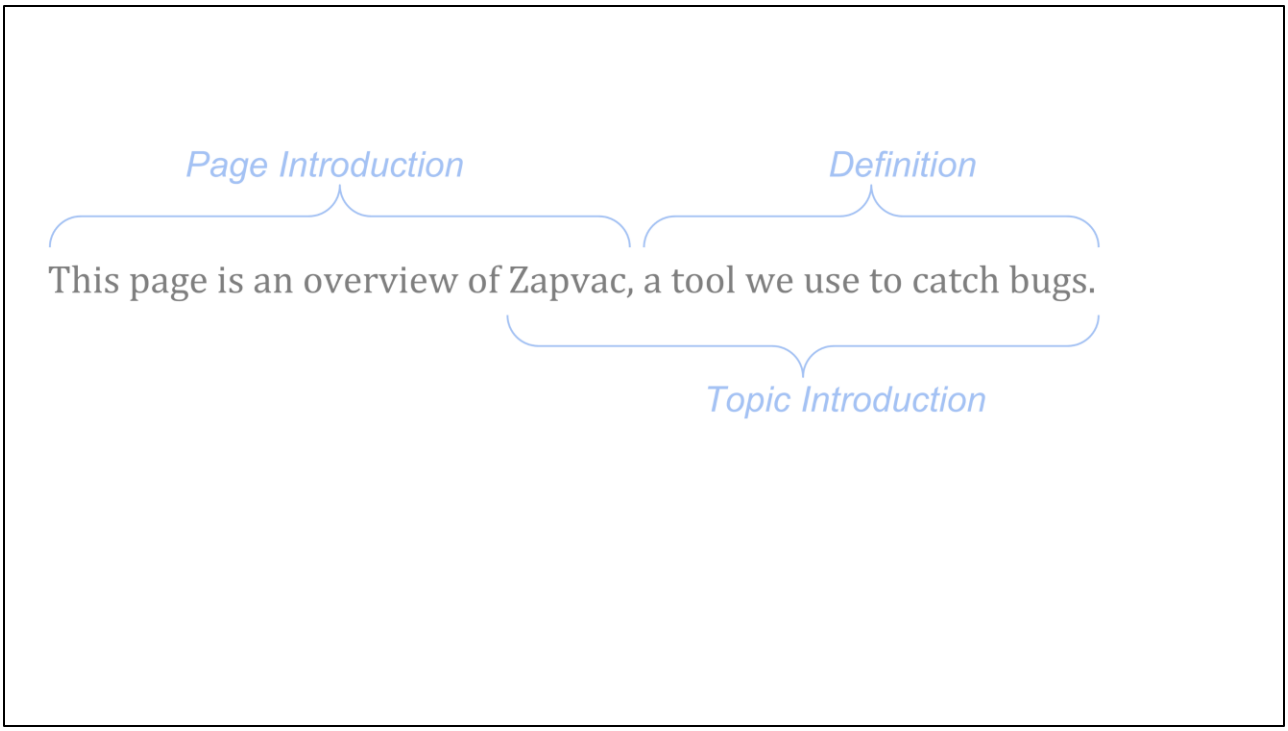
Okay, that's the end of the Structure section. We're halfway through the talk. Next up is *Presentation*, and then *Language.*



This section is about different ways to present information to the reader. You're familiar with these, so we'll move quickly.

We'll start with *introductions*, which are the second most common things we mention to engineers.

Remember that one of the most important things in reading is *context.* An introduction lets you set the context for your entire document.



"This page is an overview of Zapvac, a tool we use to catch bugs."

In about 3 seconds, this 15-word introduction tells us what *kind* of page it is, what the *topic* is, and what Zapvac is. This tells us if we're on the right page, and it tells us how to interpret what we read.

Introduce Your Page

This page is an overview of Zapvac, a tool we use to catch bugs.

~

Easy templates:

- This is a __type of doc__ on __topic__ for __audience / use-case__.
- This is a __type of doc__ on __topic__, which is __definition__.
- This is a __type of doc__ for __audience__ who __use-case__.

For most pages, a one-line intro like this is fine. It defines the *topic*, but it also tells us about the *document*. Is it an overview, a guide, a tutorial, a runbook? What's it *about*? *Who* should read it, and *why*?

What else would readers like to know before they start? Here's another example…

Introduce Your Page

This tutorial on cleaning your Zapvac takes about an hour, and requires a screwdriver. It's intended for engineers who have read the Zapvac overview.

"This tutorial on cleaning your Zapvac takes about an hour, and requires a screwdriver. It's intended for engineers who have read the Zapvac overview."

Readers appreciate this kind of meta-information.

We sometimes see wikis in which every page is good, but there is no *guidance* on which pages to read, in what order, or where it's going. *Guide your reader* with *page introductions* like this, and with *landing pages* that provide guides for specific audiences.

Remember the curse of knowledge - *you* know what your docs are about, but *they* don't. Telling readers the purpose and value of the page gives them control over their learning experience.

Now let's talk about introducing your topics.

Introduce Your Topics

Zaptrack was designed to provide clear, fast, and regular feedback to developers about their code changes with low computational expense.

This sentence introduces the *topic* of How Zaptrack Works. It lists four key ideas, marked here in color. It works like a program stub or an interface declaration, creating spaces that the author will fill later.

Research shows that the difference between experts and beginners is not just more knowledge. The knowledge of experts is better *organized*. They know categories... patterns... principles... examples… the reasons for things. All those connections make the knowledge easier to understand, and easier to remember. For a complex topic, your introduction provides an instant mental framework that helps people learn.

For simpler subjects, a short introduction is usually enough to provide *context* and *reading goals* for the topic.

The next few sentences fill in the Zaptrack *schema* a little more.

~~~

<NOTE: For more on experts vs beginners, see Chapters 1 & 2 of How People Learn.>
~~~

Introduce Your Topics

Zaptrack was designed to provide clear, fast, and regular feedback to developers about their code changes with low energy consumption.

- The feedback must be clear so that...
- The feedback must be fast because...
- The feedback must be regular because..

Zaptrack strives for low computational expense because...

Take a look. You see the text expanding on *clear… fast… regular…* and the emphasis on *low energy consumption*.

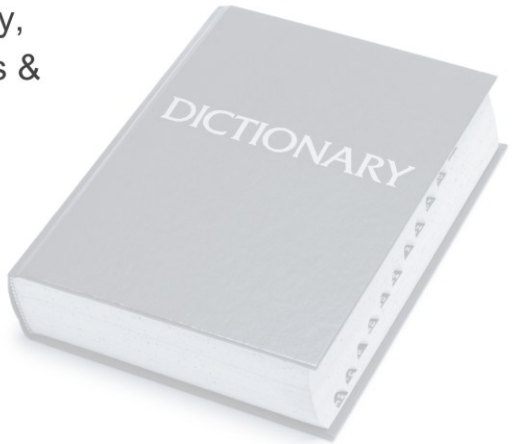
After the long introduction you see here, the doc has a separate section for each of these 4 key ideas. Consciously or unconsciously, readers are well prepared when they get there.

So. Sometimes you just say "Now let's talk about X". Sometimes you frame the big picture, like this example. Sometimes you start with a *question or problem* that sets the *context* and makes the reader curious: "Why is the sky blue?"

And sometimes you start with a *definition*.

Define Your Terms

1. **Say what it is** (use prototype or category)
2. **Differentiate** from others in the category
3. **Optionally expand** with purpose, history, principles, features, examples, strengths & weaknesses, etc.



The first time you use an important new term, *define* it. Don't make readers guess at the meaning. If you want to get fancy, *italicize* the first instance of the word (if it's not a capitalized name). That visually flags it as a new term the reader should remember.

But then define it. Don't just give examples, and don't start by saying what it *does*. Say what it *is,* using a prototype or category. It's an integer. It's a tool. It's a technique. It's an app. Then say what makes it different or special.

This is like object-oriented programming. To define a word you reference an existing class or prototype, then modify its properties.

~~~

*<NOTE: For tips on definitions, visit the Purdue University Online Writing Lab (OWL). The approach shown here is described in many sources on the Internet and consistent with "prototype theory" and "exemplar theory" in cognitive psych. Two Stanford and Duke psychologists discuss this (p. 53-62) in Made to Stick: Why Some Ideas Take Hold and Others Come Unstuck. In a famous example, the screenwriters of the movie Alien sold it to producers in three words by describing it as "*Jaws *in space". They used pre-existing mental objects instead of starting with nothing. To define more difficult words like "game" or "love", you might use Wittgenstein's "family resemblance" concept of definitions (defined in Wikipedia or in psychology terms here), in which no thing is a perfect member of a category (e.g., some birds don't fly), but there is a collective resemblance across a set of traits.>*
~~~

<NOTE: Guessing at meaning and working with unresolved definitions is cognitively (computationally) expensive for your readers.>

<NOTE: By referencing an existing word or concept, you are attaching your definition to existing schemas in your reader's mind.>

Define Your Terms

1. **Say what it is**
2. **Differentiate**
3. **Optionally expand**

✓ *Source control* is a set of tools and strategies for managing source code.

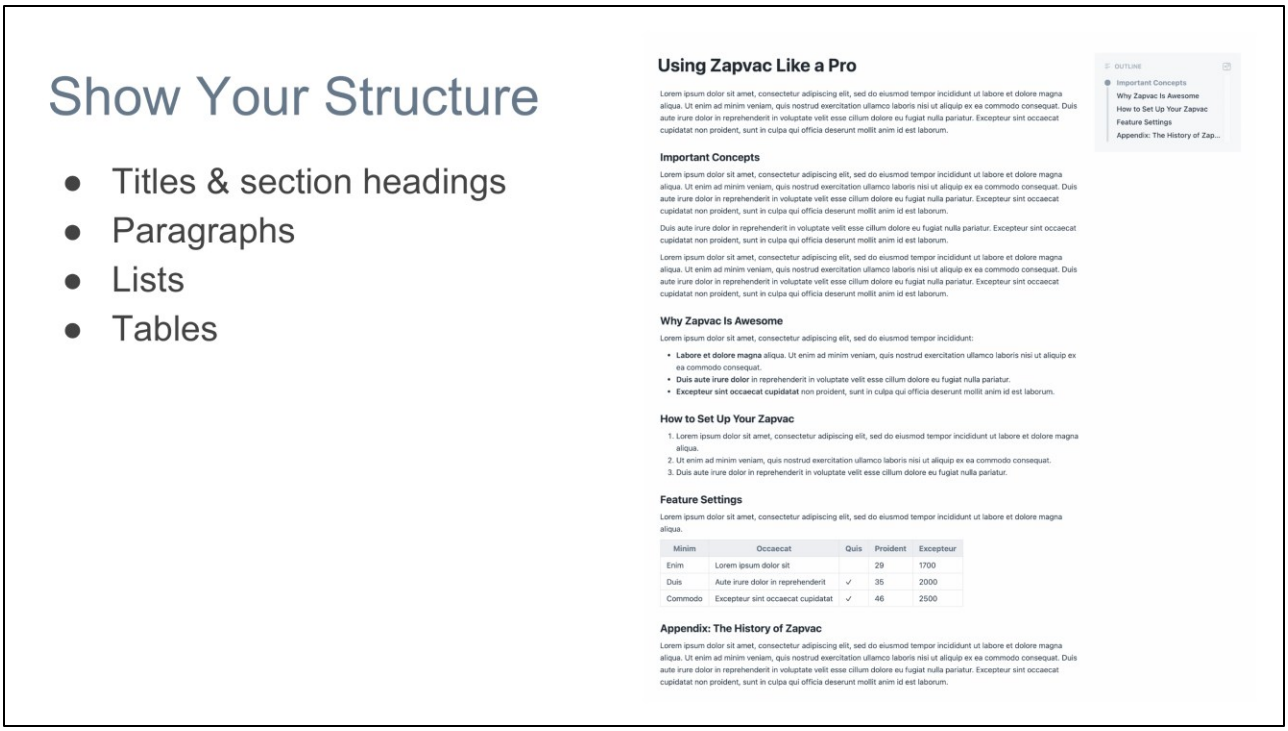
✓ A *commit* is a set of saved changes.

✓ *Zaptrack* is our continuous integration system that builds, tests, and lands your changes. It is a highly modular system that...

Here are some examples that all use this approach.

If you look for definitions as you read, you might be surprised where you see them… and where you don't.

That covers introductions and definitions, two very important features of your docs.



You worked hard on your document structure, so make sure readers can see it and get the big picture. This helps readers understand and navigate your doc.

- Use the title and headings to show structure.
- Use *H1 and H2 headings* for high contrast, a consistent visual style, and a built-in table of contents.
    - When interviewed, dyslexic and ESL engineers specifically told us that headings are helpful. Actually, that's true for *all* readers. Imagine trying to read a college textbook with no section headings!
- *Paragraphs* and *sections* are also visual clues to structure.
- *Bullet lists* visually show how many distinct items there are. Reinforce that by stating quantity and type: *"The following 3 requirements:"*, *"These 4 parameters:"*, etc.
- Use *numbered lists* only when the numbers are important, like in a list of instructions. Otherwise just use bullets.
- And use tables to clearly show information structure when you can.

~~~

<NOTE: For a sense of how much a heading can affect the reader's perceptions, watch the fun YouTube video of the Selective Attention Test, or see Chapter 5 "Reading Comprehension" of the very readable book The Reading Mind.>
~~~

Other Document Elements

- Diagrams and charts
- Screenshots
- Tables
- Code samples
- Hyperlinks

Okay, we'll end this Presentation section with tips on some more familiar elements.

Diagrams

Proprietary images removed. ☺

When using an image or table, tell your audience what it is, using a *caption* or short *introduction*. "This diagram shows the flow of data through the system."

Images should not just *repeat* what you've already said. They should add new info, or provide a good type of *redundancy* by confirming and expanding your reader's understanding.

Think in advance about whether you're doing a simple concept diagram that gets a point across quickly, or a detailed schematic that's more like reference material. Let this influence your creative decisions.

Make your diagram a good neighbor on the page - not a buzzing neon distraction that makes it hard to read the text. Use large text that fills the available space in your boxes and is easy to read. And lean toward a simple diagram that’s easy to create and maintain, rather than an expensive work of art.

Here’s a pro tip: Consider starting your diagram with a copy of another diagram that you liked.

And by the way, notice how many good diagrams use some form of *chunk & sequence*. They create *groups* with color, shapes, shaded regions, fonts, and different line styles. And they create *sequence* with arrows, alignment, and other clues.

~~~

<NOTE: For info or inspiration, you could visit the Wikipedia article on Information Design or explore Amazon starting with the work of information designer Edward Tufte.>
~~~

Charts

Proprietary images removed. ☺

Charts, sometimes called *plots* or *graphs,* are good for discussing *data* or how to *use* data, like a page about testing or performance. Sometimes sample data is a good way to explain a concept. Charts usually don't take much work, because you can create them with your data or query tools.

Screenshots

Proprietary images removed. ☺

Sometimes a screenshot is the best way to tell the story. Here are 4 tips:

1. Before you take the screen shot, shrink your browser and zoom the contents of the window to get large, readable text.
2. A detailed image becomes obsolete faster. So grab the smallest, simplest image that works. Another nice thing about small images is that they don't completely block the flow of your text or make your document seem long.
3. If you do use a large, detailed image, highlight the part you want the reader to notice, so they don't have to hunt for it.
4. Because screenshots create a maintenance burden, only use them if they provide real value.

Tables

Proprietary images removed. ☺

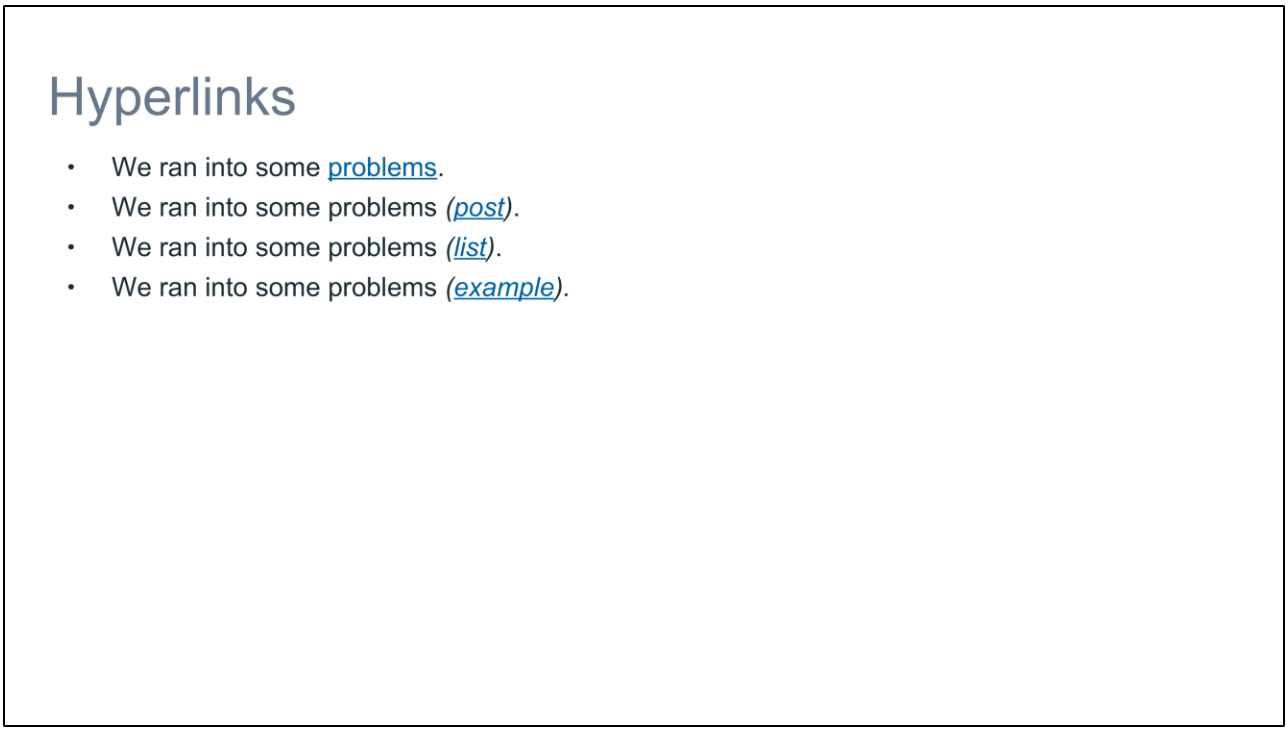
Tables communicate structure better than a sentence or paragraph. And just like images, tables engage more of the working memory, resulting in better recall. Even a small table with just 2 or 3 rows can be better than a paragraph.

You can often save time by copying a table that you like. For large tables, or to use color, try to import the table from another app.

Code Samples

Proprietary images removed. ☺

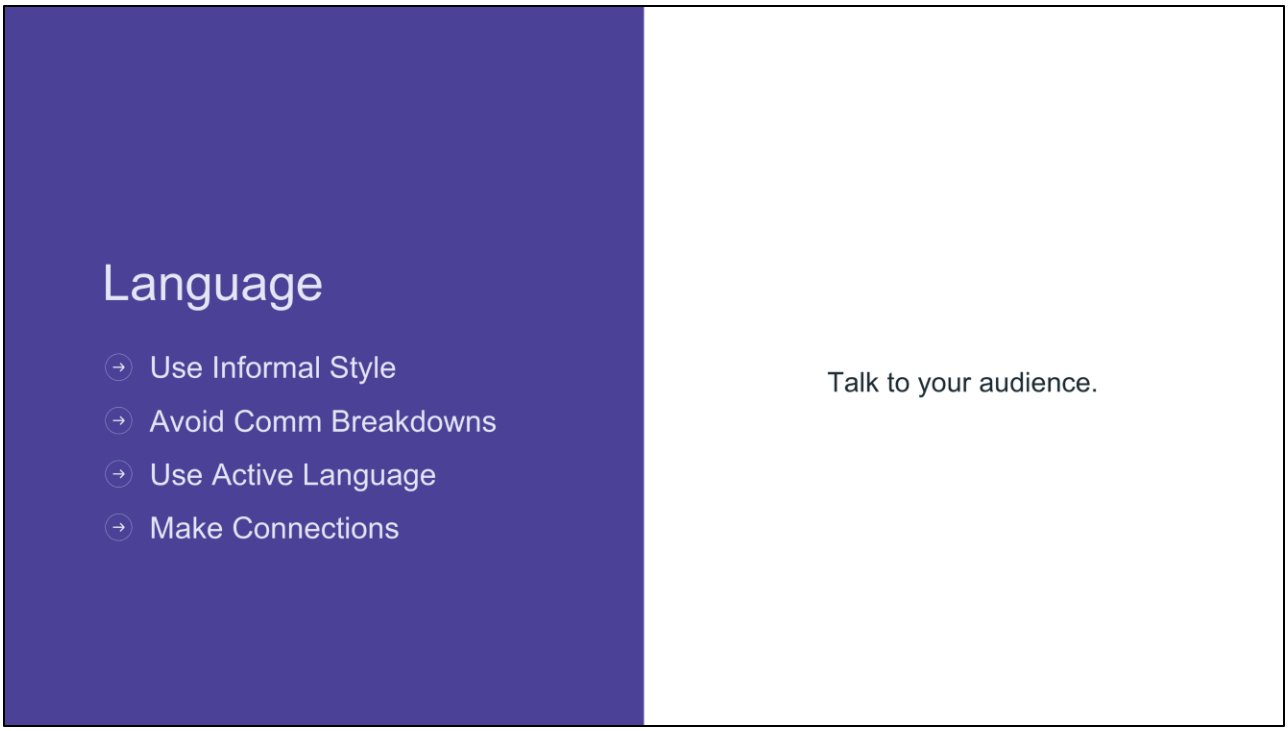
Code samples confirm and expand the reader's understanding. Some engineers would rather read code than English. A good sample is *short*, *easy to understand*, and focuses on *just the thing it demonstrates.* If explaining the logic will save the reader some time, please do explain, especially if you have a lot of readers. :)



Links can be distracting, and they break the reader's flow, especially if you send them off to other pages in the middle of a complicated explanation.

Don’t use mysterious links; make it obvious what the link is about. If they don't need it immediately, consider putting it in a list of links. Wherever possible, show the reader a reading path, not just a lot of possible things to read. Even worse, don’t send people into an endless loop of pages that all refer back to each other with no guidance.

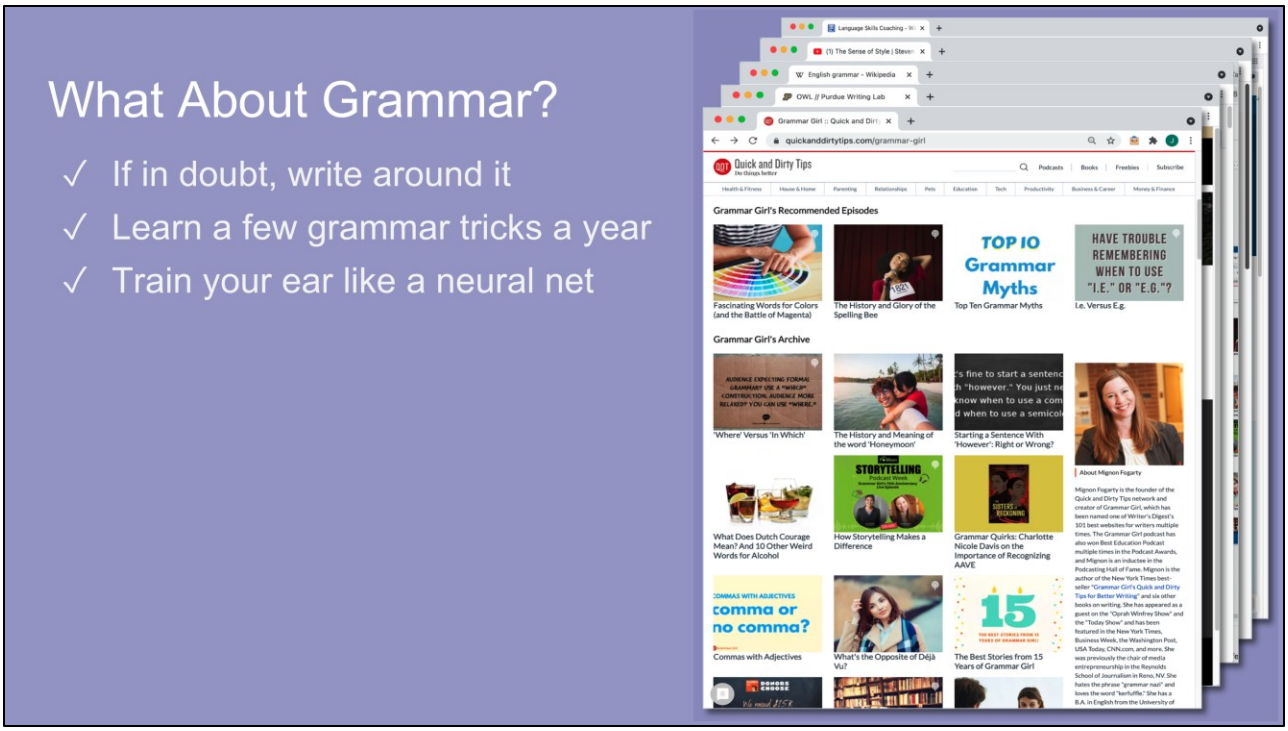
And speaking of guidance, that’s the end of this section.



Our final section is on *language* - the writing of the actual text. We'll spend a few minutes talking about four suggestions:

1. Use informal writing style
2. Don’t break the communication channel
3. Use direct, active language
4. And make connections

Some people might be wondering, *but what about grammar?*



If you want to work on your writing, there are some great resources online. But the approach in this talk does not see grammar as a big issue for internal docs. A fear of grammar should not stop any engineer from writing internal docs! Your knowledge has value, and you should share it!

To help, here are two tricks used by a lot of professional writers, and one suggestion for ESL writers.

First: If you can't remember the grammar rule for something, and don't want to look it up, just think of a different way write it. No one will ever know.

Second: A few times a year, look up a rule or a tip. Just Google something like "how to use a comma", "how to use a hyphen", or "commonly misspelled words". Watch a YouTube video, or grab some coffee and browse a grammar site. Over time this will make you a strong, confident writer.

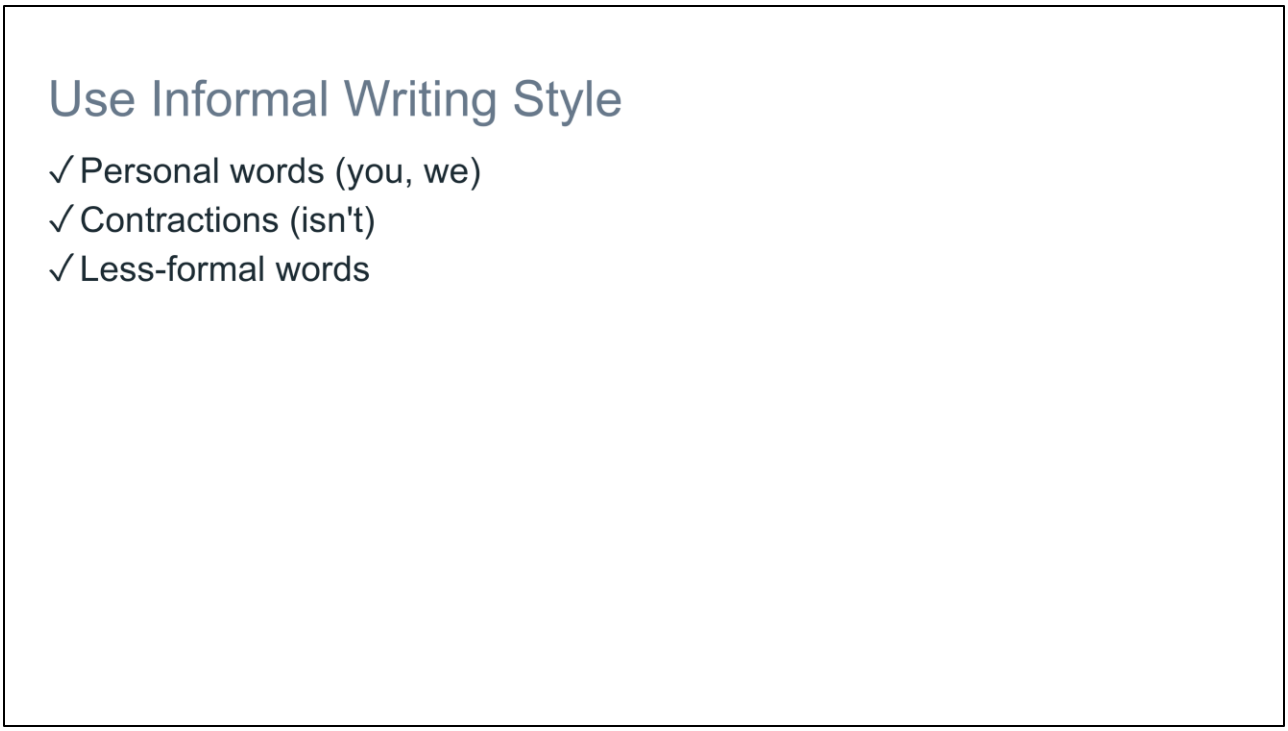
Third: If your English isn't strong, train your ear the way you would train a neural net. Listen to podcasts or, even better, watch English TV with the captions turned on. Eventually you'll *hear* when something is right or wrong.

That's it for grammar. Let’s move on to our 4 suggestions.

~~~
~~~

<NOTE: A few good resources:

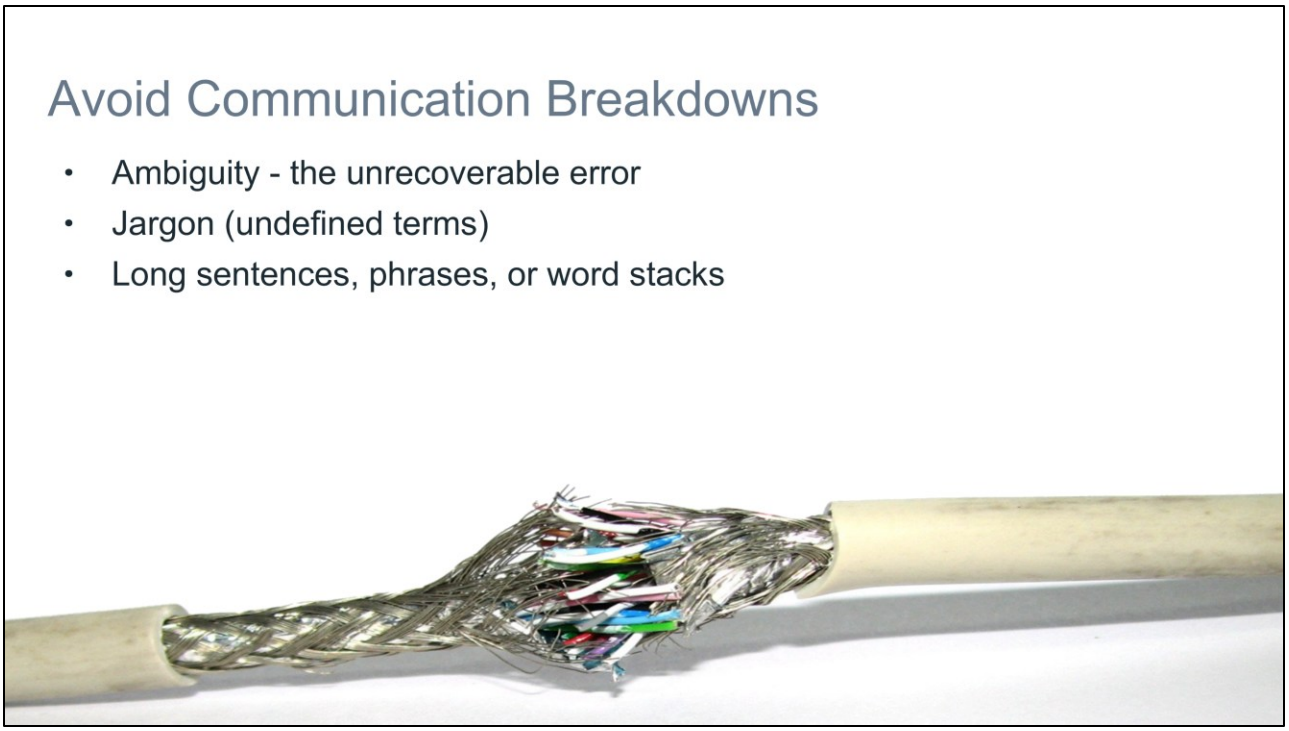
- *ESL employees at your company may have access to personal language coaching or other tools. Ask a manager or your HR department.*
- *Grammar Girl, recommended by Writer's Digest, has short articles that quickly offer clarity & insight; search "grammar girl".*
- *The Purdue University OWL (online writing lab), is free, enjoyable, and has excellent explanations and examples/quizzes; search "purdue owl" or "owl grammar".*
- *The Wikipedia article on English Grammar is comprehensive, concise, and surprisingly interesting.*
- *Steven Pinker's 1-hour Google talk on The Sense of Style uses linguistics & psychology to re-approach 21st-century grammar & style while pointing out the problems with some rusty old rules.>*



First, lean toward informal style if you can. You're probably not writing for a bank or a courtroom. Casual language is easy to write, easy to read, and easy to maintain. And studies show that *personal* words like "you" and "we" actually increase reader interest. They create a connection that improves communication. Also, it takes fewer words to talk directly to the reader than to phrase things more formally. So imagine specific readers, perhaps someone you know, and talk to them as you write.

Feel free to use simple contractions like *don't* and *isn't*, instead of *do not* and *is not* - if they feel right to you.

And use familiar, casual words. Don't use a five-dollar word when a cheap one will do.



Now, here are three things to *avoid* because they crash the communication channel. The solution in each case is to be mindful of your audience and how they will see your text.

The first one is *ambiguity*, meaning text that's unclear or confusing. A hundred years ago a US president and Supreme Court justice, William Howard Taft, said "Don't write to be understood, write so that you can't be *mis*understood." Did you get that? *Don't be misunderstood*. If you make a grammar or spelling error, the reader can usually figure out what you mean. But if your text has multiple meanings, or is not specific enough, the reader is helpless, with no way to figure it out. So *ambiguity* is probably the number one thing to avoid, worse than any grammar error.

The next crime on the list is "*jargon*", which means "insider" terms that readers may not know. So define your terms, like I just did with *jargon.*

Finally, remember that the reader's working memory has limits. A long sentence can cause a buffer overflow, forcing the reader to stop and re-read. One of the worst forms of this is a long string of modifiers, like a technical term that's six words long. Remember the Curse of Knowledge! Since you wrote that long sentence, it's easy for *you* to read, but not for your *readers*.

Use Active Language

- Short, familiar words & short sentences
- Direct, active sentences
- Be concise

Good start. Needs more gibberish.

I hope you like the cartoon. *Gibberish* is nonsense language or obscure jargon that doesn't mean anything to readers.

When we asked ESL and dyslexic engineers how to make docs more readable, they mentioned things we've already discussed: short, familiar words... short sentences and paragraphs... context... headings... diagrams... tables... code samples. But it's not just them. Researchers tell us that these things make reading easier for *everyone*.

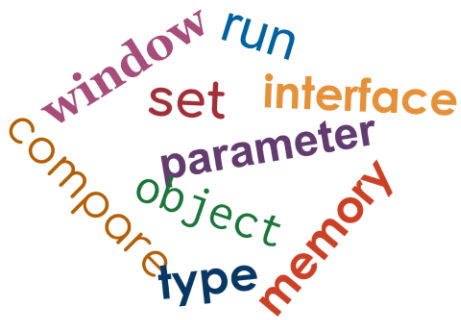
Along with *short* sentences, try to write *direct, active* sentences. In English, the simplest form of a sentence is "SVO" - subject, verb, object. "*Sophia... threw... the ball*." Hear it? Subject, verb, object. "Sophia threw the ball." SVO is shorter and easier to unpack than the long, backward, *passive* form, which is: "The ball *was* thrown *by* Sophia." SVO is faster and more direct. Now, sometimes it's good to capture a big idea in one long sentence. But try to prefer short, direct, active SVO sentences.

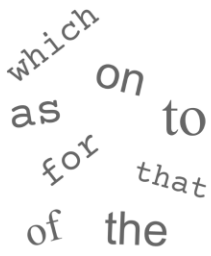
~~~

<NOTE: The most popular readability metrics are based on word length and sentence length; see the article Surface Readability for more info, or visit Readable.com or Wikipedia. A useful and fun resource is The A-Z of Alternative Words, which lists simple alternatives to many long words or expressions. For amazing ideas like placing context at the start of a sentence, see the 1990 paper The Science of Scientific Writing, by George Gopen and Judith Swan. For more on the pros and cons of passive sentences, including how to use them to link the end of one sentence to
~~~

the start of the next, see Stephen Pinker's 2014 book The Sense of Style or his Google talk about it on YouTube.>

Use Active Language

- Short, familiar words & short sentences
- Direct, active sentences
- Be concise





Engineers might be interested in a concept called *lexical density* - the ratio of *semantic* words, like *compiler* or *performance*, to words that are more about *syntax*, such as *of, the, that,* or *to*. If your text has a high percentage of those syntax words, it feels like it's not going anywhere. Readers like writing that gets to the point. In physics, we'd talk about *velocity,* or maybe *work*. In information theory, we'd talk about creating *entropy* or surprise.

~~~

<NOTE: To experiment with lexical density and other fun metrics, try analyzemywriting.com.>
~~~

Use Active Language

- Short, familiar words & short sentences
- Direct, active sentences
- Be concise

Are you ~~even~~ being serious right now?
Are you being serious ~~right~~ now?
Are you being serious ~~now~~?
Are you ~~being~~ serious?
~~Are you~~ serious?
~~Seriously~~?
Really?

Be *concise*. Writing concisely is hard; you have to write it, then trim it, then trim it again. Take out words that don't add information, and take out anything that's repetitive. Don't spend all day on it, but make an effort.

And try to write simple, direct, active sentences. You don't need all *active* sentences, or all *short* sentences. A good mix will delight your readers. But be direct and active. Throw the ball.

These techniques give your busy readers a sense of forward motion. And so do the tricks on the next slide, our last Language slide.

Make Connections

- Use connecting words: *first, and, or, if, but, also, then, so, with, always, never, since, because, before, instead, however, therefore…*
- Use story: goals, problems, themes, solutions, lessons

Make connections. It increases interest, reduces skipping, and improves memory. And it does one other important thing: it communicates *insight*.

So here are two tricks:

First, use *connecting words*. They show relationships: similarity, opposition, cause, effect, origin, grouping, sequence, and so on. They add meaning, and create a sense of forward motion. *Without* connecting words, you sound like a robot reading a list of unrelated facts. *With* them, your ideas are logically connected, and the reader feels carried from one sentence to the next.

Second, use *story!* Humans are wired for story and problems. We spend hours watching movies about people with problems. We love problems! And the conflict or objective in a story is the ultimate reading goal: *We built this tool to solve the problem of… The best part of this solution is… But one major weakness is…* Just that much story or context engages your reader, makes meaningful connections, and helps create strong, memorable schemas.

We're almost through, but maybe you've been wondering something all this time.

What If I’m a Terrible Writer?

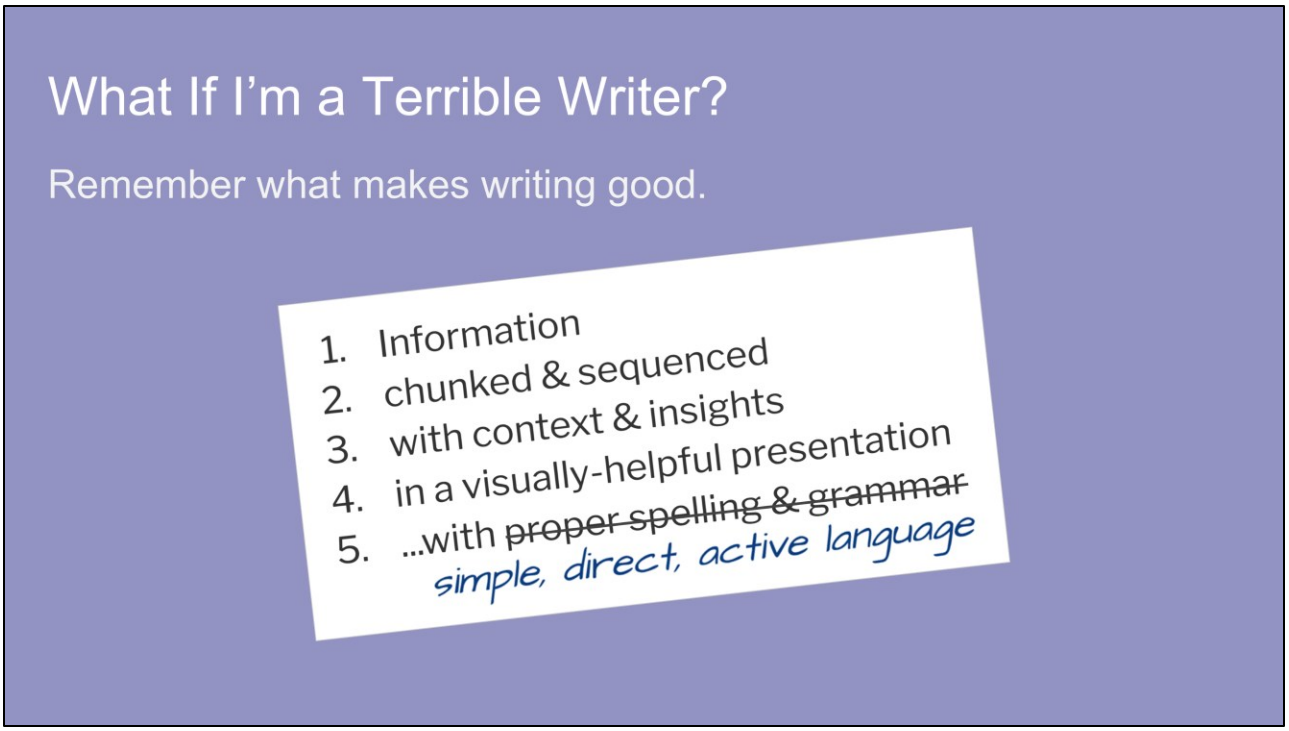
"What if I'm a terrible writer?"

What If I’m a Terrible Writer?

Remember what makes writing good.

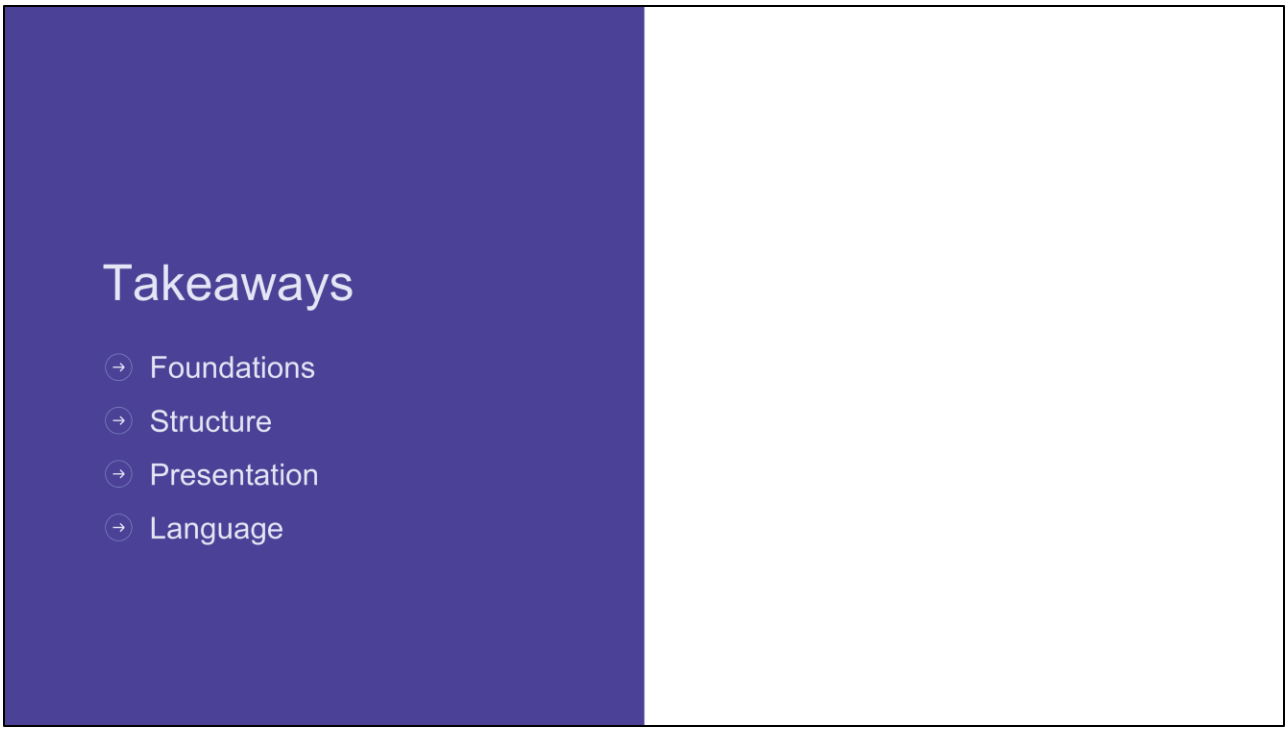
That’s probably not going to happen.

Remember what we said makes writing good.

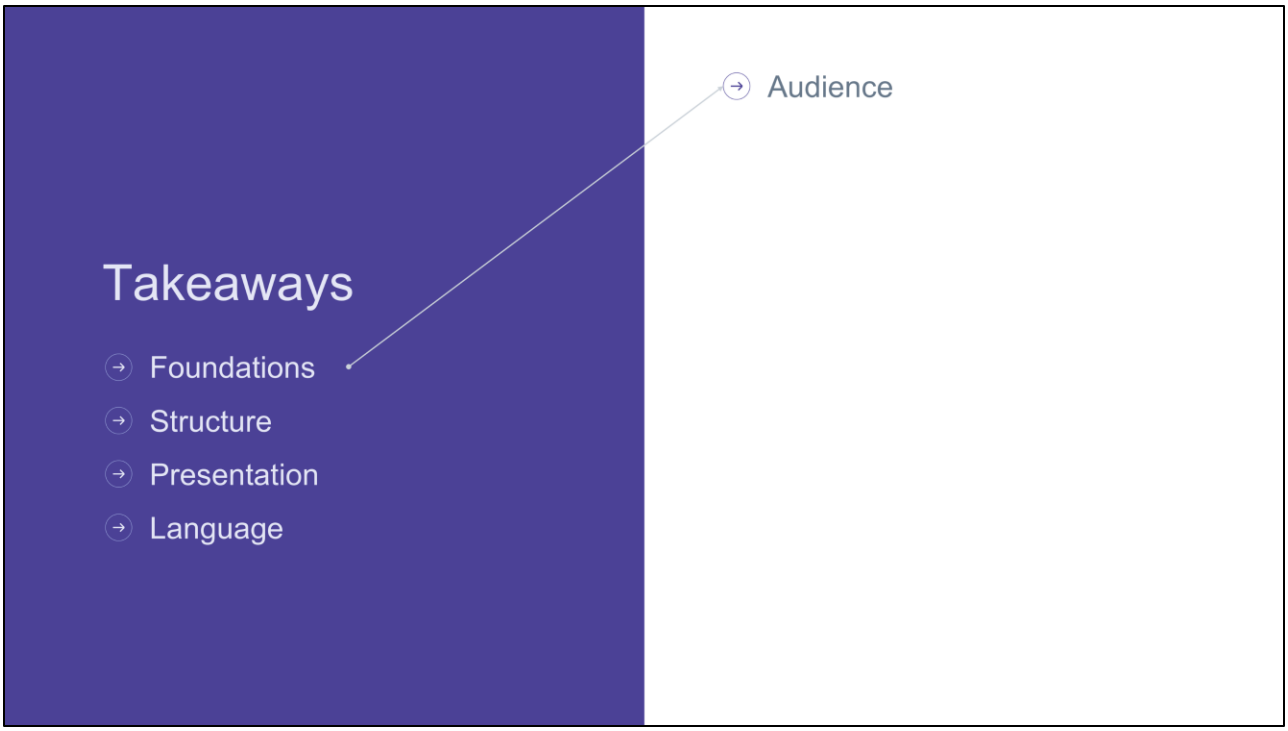


Accurate information... in a logical structure... with context... and insight... on a page with headings, introductions, and sometimes lists, tables, examples, images...

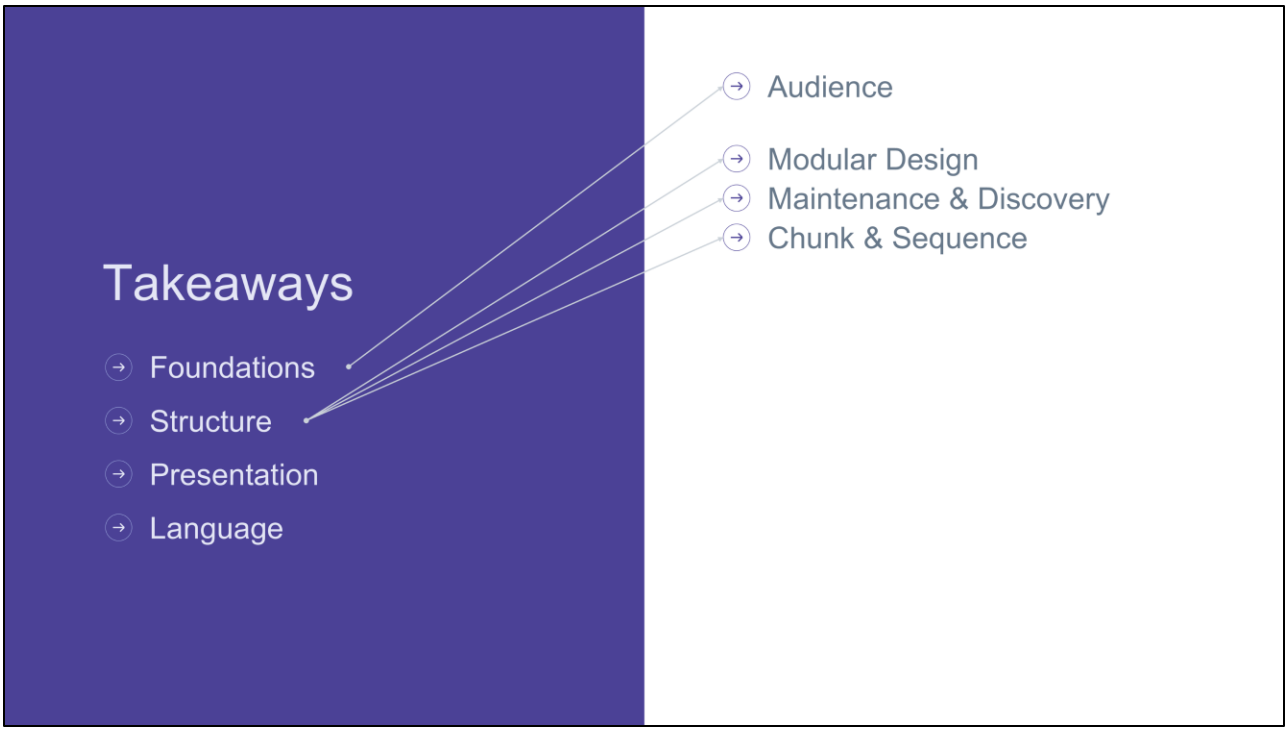
and simple, direct, active language.



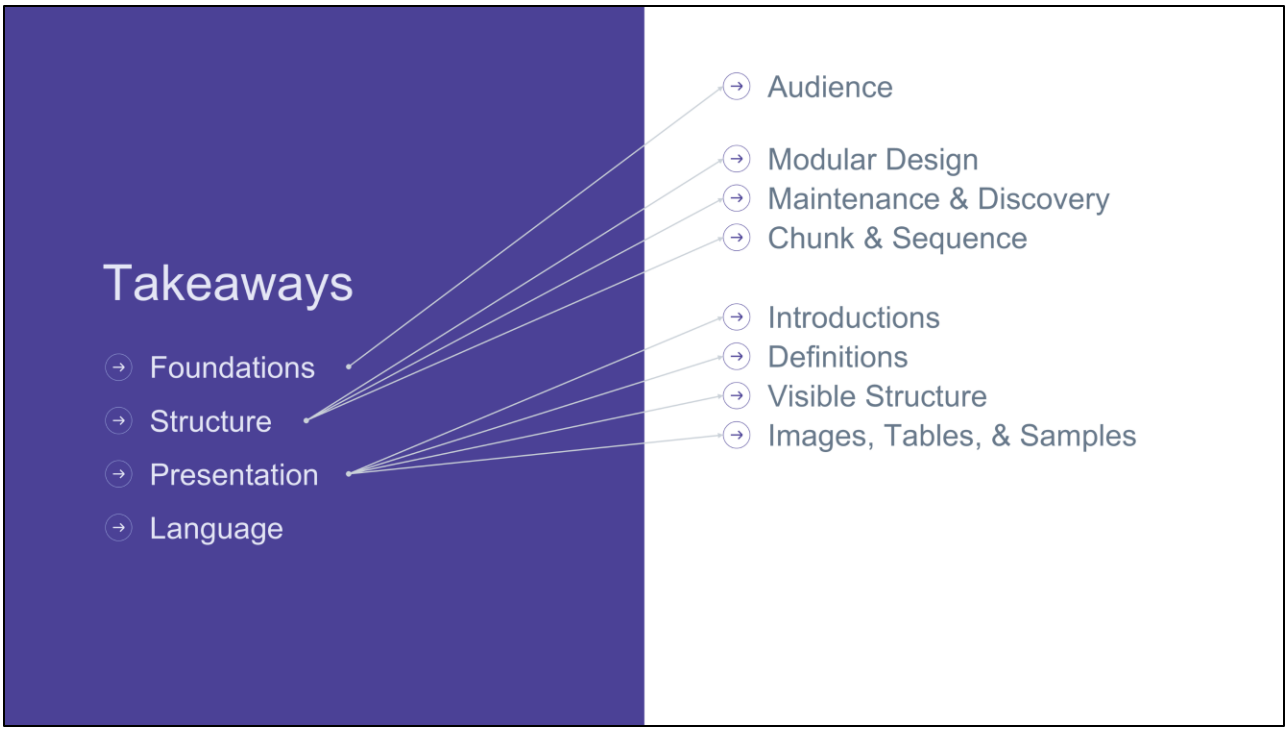
Okay, let's recap our writing stack.



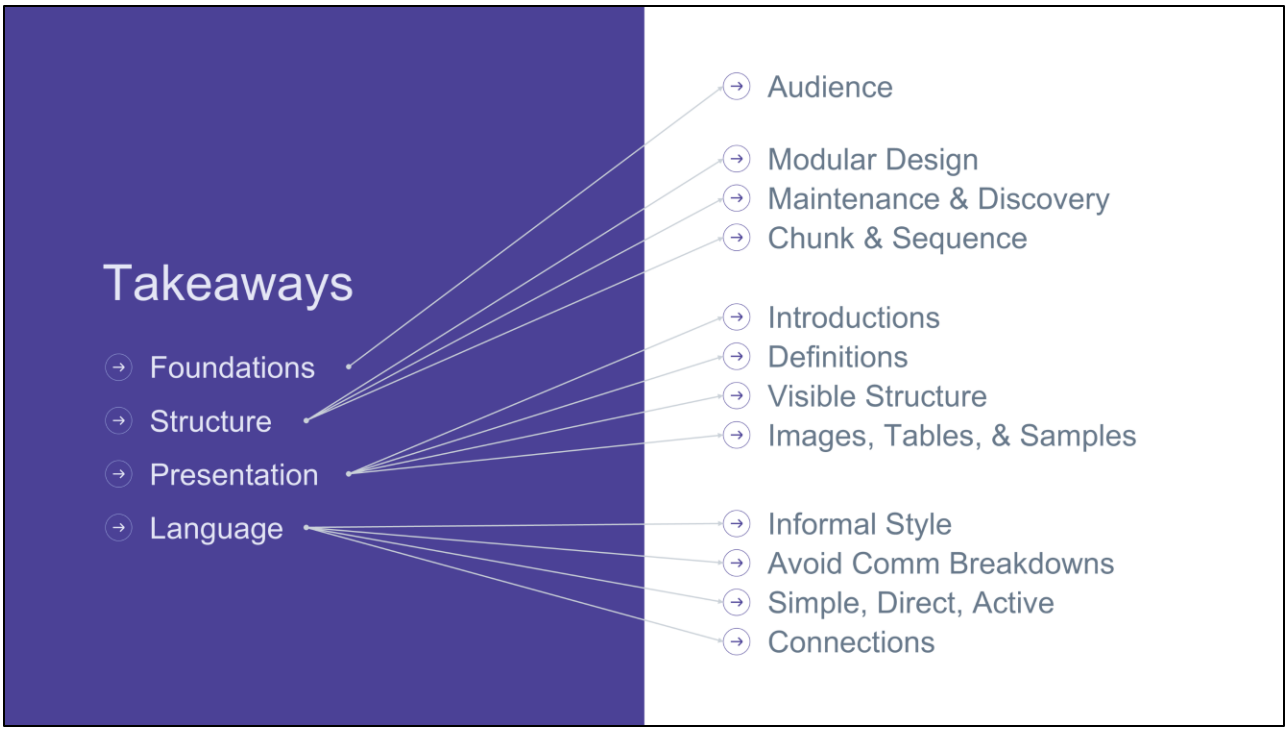
- Think about audience & use-case.



- Build modular docs. Chunk and sequence everything.
- Think about maintenance and discovery.



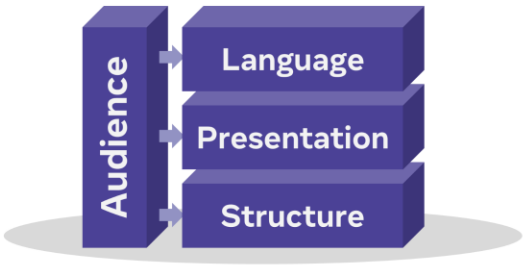
- Introduce your page and your topics.
- Define terms.
- Make structure visible.
- Think about effective presentation formats.



- Avoid communication breakdowns like ambiguity, jargon, and long sentences.
- Speak simple, direct, active language to your audience.
- And make connections for your readers.

That's it.

Questions?



<END OF TALK>