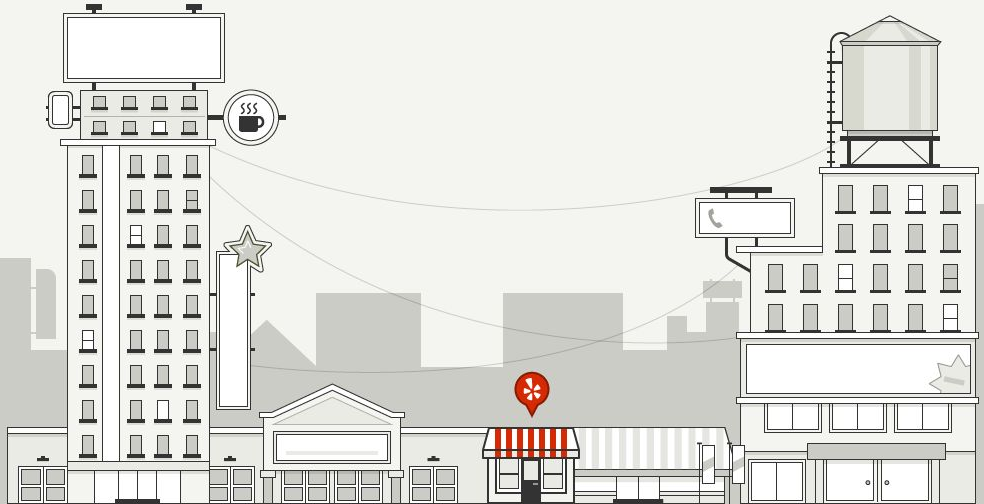
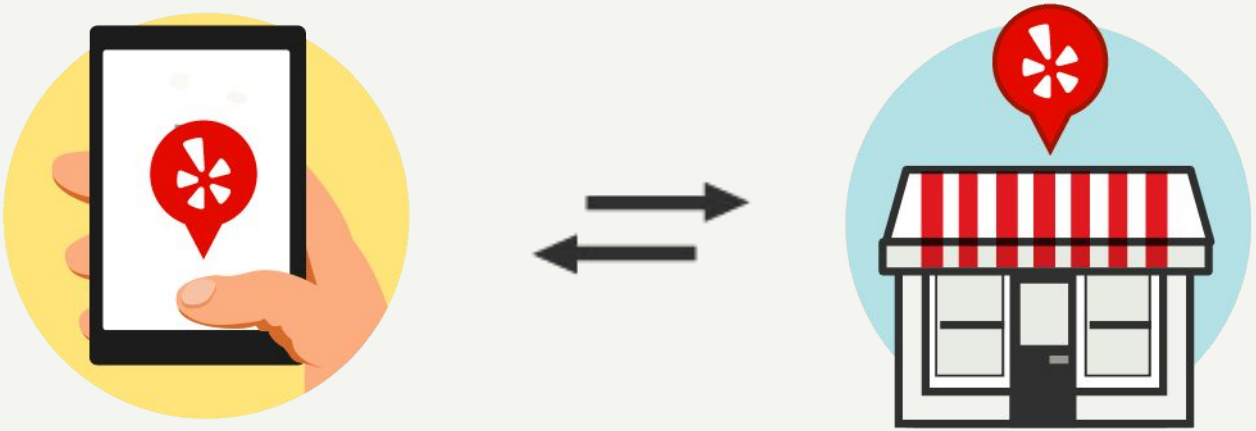
The 7 Deadly Sins of Documentation

Chastity Blackwell
cblkwell@yelp.com / @Black_Isis



Yelp's Mission

Connecting people with great local businesses.



Who am I and why do I care?



Who thinks they have good docs?



Why do you need good documentation?

- Onboarding
- Institutional Knowledge
- Incident Response
- Inclusiveness
- Self-Perpetuation



Why don't we have good docs?



Does any of this sound familiar?



You've got a doc that explains everything about that service and you're sure the info you need to solve this incident is in there -- somewhere.



"You can try looking for that in the wiki, or maybe it's in the Google Docs repo. Oh, and I've got some notes in my home directory, and I think I saw some email about that a while ago."



You search your wiki and find three separate docs on how this service works, two of which contradict each other entirely, and none of which have been updated in the last year.



You've got a 500 line Puppet manifest to handle this service...with no comments. Or comments that reference tickets from two ticketing systems ago.



The new person on the team spends the first month asking what bits of internal jargon mean.



You have a bunch of archived presentations that discuss all sorts of infrastructure components, but you have no idea how up-to-date they are because you haven't had time to watch them in ages.



The Root of All Evil: Deprioritization



Documentation is seen as "extra" work.



Why don't we prioritize documentation?



What happens?

- No one has time to go back, or when they do, they don't remember why things were done
- There's a tendency to see documentation as "lesser work" by engineers and management -- quality suffers
- Often relegated to new or junior members of a team, who lack context necessary to write good docs
- No one learns how to improve their writing



Documentation isn't extra work.

It's necessary work.



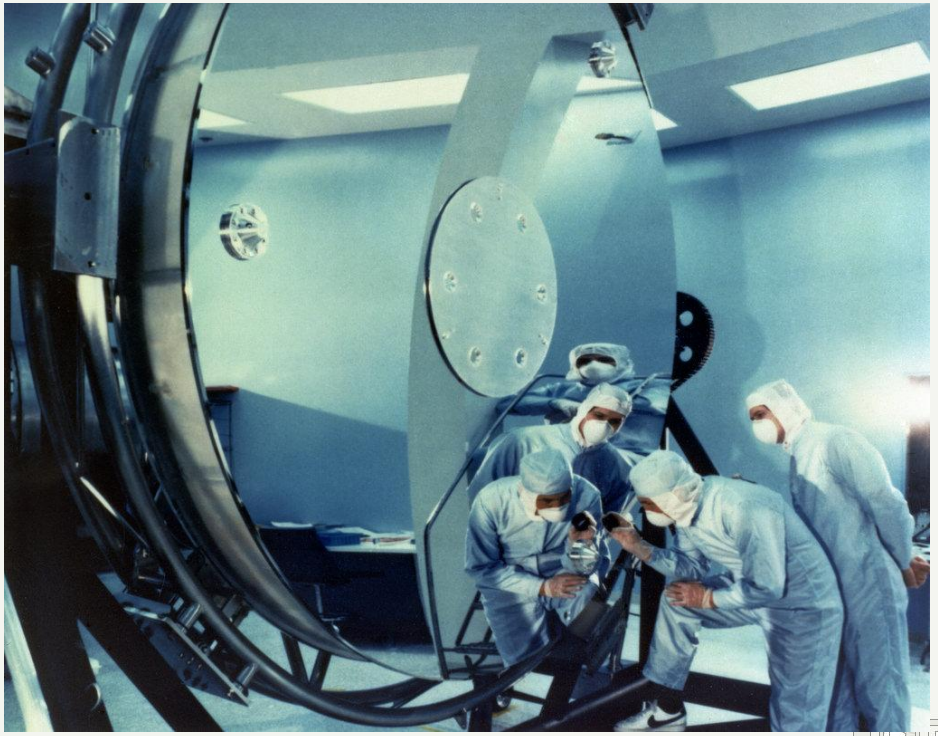
Documentation as a deliverable



Create documentation standards and style guides



Have a real review process



Everyone needs to pitch in



The Six Subsequent Sins

Burying the Lede
Documentation Overgrowth
Repository Overload
Comment Neglect
Jargon Overuse
Video Addiction



Burying the Lede,
or the Master of None



- When in an incident, you need immediately useful runbooks
- When you're trying to learn how a system works, you need a document with a thorough description of how it works and what it interacts with
- Too often we have one or the other -- or we try to do both with a single document



You wouldn't describe how to fix a tire by starting with a car factory.

You wouldn't describe how a car is built by starting with how to fix a tire.



Runbooks

- Address specific questions; make sure alerts link directly to diagnostic and remediation steps
- Use inverted pyramid format; put the most important information or more common questions at the top
- Make sure your example commands will be benign
- Have explanations of why you're doing this, but keep them short -- one or two sentences
- Make sure the steps in the runbook actually work!



ohshitgit.com

Oh shit, git!

Git is hard: screwing up is easy, and figuring out how to fix your mistakes is fucking impossible. Git documentation has this chicken and egg problem where you can't search for how to get yourself out of a mess, unless you *already know the name of the thing you need to know about* in order to fix your problem.

So here are some bad situations I've gotten myself into, and how I eventually got myself out of them *in plain english**.

Oh shit, I did something terribly wrong, please tell me git has a magic time machine!?!

```
git reflog
# you will see a list of every thing you've done in git, across all bra
# each one has an index HEAD@{index}
# find the one before you broke everything
git reset HEAD@{index}
# magic time machine
```

You can use this to get back stuff you accidentally deleted, or just to remove some stuff you tried that broke the repo, or to recover after a bad merge, or just to go back to a time when things actually worked. I use `reflog` A LOT. Mega hat tip to the many many many many many people who suggested adding it!



Technical Documentation

- Be sure to include the context of why things were built this way, not just how
- Provide concrete examples for how things function
- Know your audience
- Don't rely on autogenerated documentation
- Be wary of sending people down a rabbit hole
- Make sure what people understand is what you meant to say



Repository Overload



Water, water everywhere...

- Docs in a wiki, in a Google Doc, in a git repo...
- How does someone know where to find the docs they need, or where to put new docs they write?
- Especially a problem in a crisis -- incidents will last longer, SMEs will have to be brought in more often, burnout becomes a bigger danger.



Putting Things in Order

- Keep docs in one place as much as possible, at least for each type of documentation; multiple places to edit causes drift and confusion
- Make sure whatever repository you use can track changes and is searchable
- Provide portals that give easy access points into docs, especially one that surfaces critical or top-level docs
- Don't make the mistake of using email, chat logs or tickets as primary sources of documentation



Documentation Overgrowth



Outdated, conflicting, or misleading

- Documentation not updated to reflect new realities
- Keeping documentation around for "historical purposes" clutters searches and can slow incident response
- Old, incorrect docs lie around, which is especially bad for new or inexperienced team members



Pruning The Garden

- Include documentation in project goals
- Instead of keeping old documentation around, pull out what is still useful and incorporate it into new docs, discard the legacy doc
- If you need to keep them, make sure archived docs will not show up in searches



Pruning the Garden

The best way to keep your docs up to date is to use and update them regularly.



Comment Neglect



"The code is the documentation"

- Even good code does not show the context of how decisions were made about why the code was written that way ("Is it supposed to work this way?")
- Tendency to assume your code is easy to read and understand -- if you've been working on it for weeks, it seems intuitive, even if it's not
- Someone else may not have as good an understanding of the language or functionality of the code



Here's some better comments:

```
# See this page for context on why we use this
# method: http://ourwiki.com/thing

# Optimally we'd hit the API, but we got throttled
# last time we did that, so we look at the cached
# results instead.

# This configuration file sets up the regional
# endpoint for us-west; see gslb-prod.tf for how
# it ties in to the overall config and weighting.
```

@Black_Isis

Context is King

- Make sure people can understand what your code is *supposed* to be doing and why
- Make sure comments or documentation on the project describe the context for how decisions were made about how to write certain functions
- Documentation should include concrete examples of the intended behavior
- Is there someone on your team who's never looked at that code before? Can they tell what it is for?



Jargon Overuse



I don't get it...

- Especially in young organizations, a close-knit engineering staff can use a lot of specific jargon or shorthand.
- New people need to learn the new lexicon, obscuring relationships between components, causing confusion, and keeping people in the "out" group.
- Do you want impostor syndrome? Because that's how you get impostor syndrome.



Bridging the Gap

- Avoid jargon if possible (especially if there's a more general term for it)
- Keep the audience for your documentation in mind
- When unavoidable, provide a glossary and plenty of context so it can still be understood
- Create an inclusive environment where people feel safe asking "stupid" questions



Video Addiction



Pivoting to Video?

- Video is more engaging than text (maybe)
- Can be easier to present and explain complex topics in a video than in a flat document
- Video can be more memorable than a flat document



But...

- The big problem with relying video (and graphics to a lesser extent) is that they cannot easily be searched
- If there's no content marks, you've got to watch the whole thing to find what you need
- Much harder to edit if the content needs to be updated, especially video
- They're also not great from an accessibility point of view -- screenreaders don't work well with them, and videos need subtitles for hearing-impaired folks



Use Videos and Graphics Responsibly

- Don't use video as your primary documentation method
- Videos are great for things that need to be consumed end-to-end and are updated infrequently, like onboarding
- Make sure graphics have alt-text, don't cram in too many
- Make sure videos have timestamps for content and subtitles (and consider transcripts)



Keeping Free From Sin



- Have runbooks and have in-depth technical docs, but don't try to do both in same place.
- Store your docs in one or two repositories that keep a history, are easily searchable, and easily updated
- Update them frequently and don't clutter the repo with old docs
- Provide plenty of context in your code comments
- Avoid jargon where you can, define it where you can't
- Be wary of overusing video and graphics



Documentation isn't extra work, it's necessary work.




We're Hiring!
www.yelp.com/careers/





fb.com/YelpEngineers



@YelpEngineering



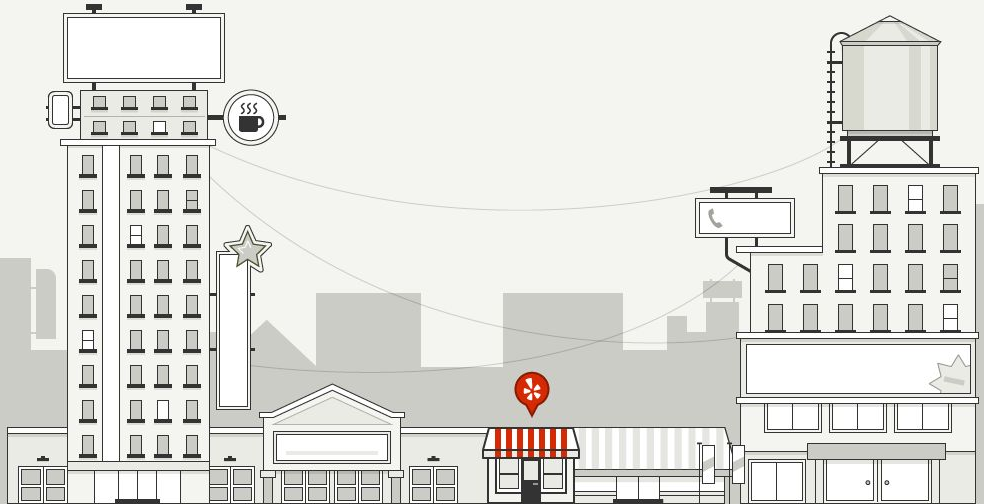
engineeringblog.yelp.com



github.com/yelp

The 7 Deadly Sins of Documentation

Chastity Blackwell
cblkwell@yelp.com / @Black_Isis



References and Inspiration

- 10 Tips For Making Your Documentation Crystal Clear (Ben Cotton, 2016)
- Code Tells You How, Comments Tell You Why (Jeff Atwood, 2006)
- Github Open Source Survey 2017 (Github, 2017)
- Oh shit, git! (Katie Sylor-Miller)
- Scalable Meatfrastructure (Alice Goldfuss, LISA 2015)
- Traps and Cookies: A Mystery Package From Your Former Self (Tanya Reilly, LISA 2016)



References and Inspiration (cont)

- What to Write/Technical Style (Jacob Kaplan-Moss, 2009)



Image Credits

Super Guppy (https://commons.wikimedia.org/wiki/File:Orion_Heat_Shield_Transported_Aboard_Super_Guppy_Plane.jpg)
Semiotic Standard (https://en.wikipedia.org/wiki/Semiotic_Standard)
Hubble Mirror Inspection (https://www.flickr.com/photos/nasahubble/27712255473)
Sandbagging (https://commons.wikimedia.org/wiki/File:FEMA_-_40289_-_Sand_bagging_operation_at_the_Fargo_Dome_in_North_Dakota.jpg)

