Write the Docs Documentation

Release 1.0

Nathan Yergler

May 08, 2014

Notes from Write the Docs 2014, May 5-6, Portland, OR.

See also: http://write-the-docs-2013-notes.rtfd.org for last year's notes.

CHAPTER 1

Monday

1.1 Morning

1.1.1 Welcome to Write the Docs

Authors Eric Holscher

Time 09:00 - 09:20

Session

Link

Write the Docs is a community conference; this means it isn't about profit, it's about getting people in a room to communicate with each other.

This is version 1.0; last year was version 0, we did one in Budapest, sponsored by Prezi (it didn't count for versioning). Budapest was amazing, and there was a lightning talk involving bag pipes at 3pm in someone's office (apparently they were cool).

Oh yeah, we still don't know what we're doing.

Your job: make friends, learn something, have fun. You're sitting at round tables, so now you have friends. We get along with one another by being welcoming, open, supportive, and positive. This is a professional conference for many of you, but it's also about building a commmunity.

The agenda today is wide-ranging, and documentarian bound. There are writers, project managers, and developers here, and we hope everyone will be stretched in at least one direction.

This year there's dedicated space for the "hallway track", on the second floor. It's free space in the morning if you just want to get away, and after lunch there will be a couple talks, and then "unconference".

1.1.2 Flow: A Permaculture Approach to Documentation Projects

Authors

18. (a) Homer Christensen

Time 9:20 - 10:00

Session http://docs.writethedocs.org/2014/na/talks/#r-n-homer-christensen-flow-a-permaculture-approach-to-documentation-projects

Link

Technical writer and instructional designer, and a practicing permie. [Lots of people konw what that is, but I have no clue.] Over the past three years he's replaced his lawn with a food forest, and signle-sourced training and documentation materials for a long term project with California Dept of Corrections.

Permaculture was first described by Bill Mollison and David Holmgren in 1978 and 1988. Permaculture is a philosophy of "working with, rather than against, nature"; "protracted and thoughtful observation" of systems to see how they evolve. "The solutions are embarassingly simple".

Examples:

- Lawns
- Gardens
- Greening a desert near Sea of Galilea
- Christensen's food forest
- City Repair: Portland based group reclaiming neighborhood public intersections to build community

Principles:

- Work with the elements
- Diversity gives stability
- The problem is the solution
- Make the least change for th greatest effect
- No limit ot the richness of design

"Everything old is new again"

http://permacultureprinciples.com

Documentation is ripe for this: you want to develop a system that can will be easily maintained and fruitful for those who come after you.

- Observe
- Design
- Evolve

Observation always comes first, and is about receiving information and understanding the environment around you. You observe without preconceived ideas, and with an open mind. If you feel like something is missing, you observe some more.

When you're observing a project, you also need to observe the surrounding environemnt. What's the corporate culture? What's the duration? What's the desired outcome/yield? Who's available, and what are their skills, abilities and natural inclinations? How can things in the environment (the land, the people, etc) be combined for the most beneficial outcome?

Design is where most of the work actually takes place. What are the zones of access? What's the smallest change that will creat the largest effect? And what direction are energies going in? It's easier to work with the flow than against it. In permaculture, the edge – where the forest meets the plain, the plain meets the ocean – is where the action is; there's nutrient exchange, there's biodiversity. So when we think about applying these principles to projects and documentation, think about where the edges are between different people roles, etc. If you can increase the size of your edge, you can increase the diversity on your project.

"The importance of collaborative porcesses is often ignored because of the urgnecy of direct action." – Telford

And the process has to evolve to continue to be successful. Observing lessons learned, measuring and evaluting outcomes, etc. What are the documentation *opportunities*?

1.1.3 Communities Are Awesome

Authors Ali Spivak

Time 10:00 - 10:20

Session http://docs.writethedocs.org/2014/na/talks/#ali-spivak-communities-are-awesome

Link

Works for Mozilla, which some people may not be aware is a non-profit, mission-driven company. Here today to talk about community. As a non-profit, Mozilla relies on their "huge" network of volunteers – their community – to actually make things happen. Ali is responsible for the Mozilla Developer Network, the MDN. MDN is a wiki that the community uses to document "everything that matters to web developers", as well as information about Mozilla products. That's a lot of of information to document, and MDN is really big. MDN has about 2MM users per month, 35 languages, and 11K articles. Things don't stay the same for long, so there's a lot of change to manage. MDN has 5 paid writers, althogh they do a lot more than writing: every one of them works with and for the community.

The community is critical to MDN's success. The community writes, edits, fixes typos, and translates content (user experience for the site, as well as articles). MDN also holds doc sprints where they get a group of writers together for a day or more, and everyone writes.

But more important than all this work, the community provides MDN with an amazing amount of diversity and perspectives. People in San Francisco may not think of low bandwidth as a topic to document, but the community of writers, programmers, and users around the world help push that forward as a topic that's important.

It's also interesting that MDN, for being so open, gets a very small amount of spam and malicious edits. Part of that may be a result of the fact that it's somewhat of a niche site. But it also seems like people treat the site with an amount of respect and responsibility to the site and their *peers*.

So you have this community; how do you get them to do what you want? You don't. Their not minions. They're a special herd of awesome cats. They're partners.

"The idea of commnunity may simple come down to supporting and interacting positively with other individuals who share a vested interest."

So why do people spend their time doing something they're not compensated for, that's not their "job"? It turns out that intrinsic motivation is really important to lots of people: autonomy, gaining mastery, and [something else I didn't write down in time] provide people with a lot of satisfaction, and volunteering on MDN helps them achieve that.

You also see this in things like Burning Man, huge events that are volunteer run. One of their principles is **Participation**. Everyone is expected to participate to help make the event successful. Mozilla has principles, as well, and they also emphasize transparency, community, and engagement.

It turns out it matters less what the principles are, you just need to have them, so that your community understands there's something bigger, that they're not just minions.

1.1.4 The New Sheriff in Town: Bringing Documentation Out of Chaos

Authors Heidi Waterhouse

Time 10:20 - 10:40

Session http://docs.writethedocs.org/2014/na/talks/#heidi-waterhouse-the-new-sheriff-in-town-bringing-documentation-out-of-chaos

Link

Technical documentarian of almost 20 years, and here to talk about the chaotic environment and how to bring some order out of it. By the time people get to this point in their career, most people have specialized in something; she's specialized in coming into organizations as their first documentarian.

So how does this work? Well first, you need a Star. And the only Star you're going to get is that you got a job: you convinced a hiring manager that you could do the job, that you could handle their thorny problems. And no one is going to give you anything else. So you have to get over your imposter syndrome.

And then you set up shop. When you roll into town, people will show you around. But you probably won't remember any of that, so make your own map/seating chart of people and documentation. So that when someone says, "Hey, Janet knows about SQL, ask here," you're able to approach Janet confidently. And there's almost certainly existing documentation, no matter how ill maintained. And while you're waiting on your laptop, provisioning, whatever, get to know the neighborhood: read the docs of your competition, of others in your domain. Understand the lay of the land.

Once you start writing, you have no time for frills. You need to draw fast. Polish and precision (formatting, localization, etc) will happen between emergencies. Deliver early, deliver often, or talk about why you can't deliver. Early/often delivery helps demonstrate why this documentation thing is important. The polish and precision is important, but it doesn't matter if you don't put out the fire first.

And now you're going to save the townspeople: find the biggest pain point you can address and take care of it. Maybe it's just a PDF form that customer support doesn't know is possible. It helps others, gets them on your side, *and* quiets your internal imposter syndrome. You also need to give others a way to ask questions and communicate with you. A bug tracker is a great way to handle this, and gives you a punch list to work down on days when writing feels like too much.

You'll also often find scorpions: documentation hoarded in inaccessible tools, or a disorganized manner. The people who are hoarding are vigilantes: they care about documentation, but they're operating outside the law, so you want to deputize them and bring them into the "official" documentation project.

Finally, it's important to build infrastructure. Once the fires are out, it's important to provide some structure for continued success. This means templates, release notes, etc; make sure that documentation is part of the process for new code, new features, etc. That ensures that it becomes part of the long term success of the organization.

1.1.5 Ignorance is Strength

Authors Amalia Hawkins

Time 11:00 - 11:40

Session http://docs.writethedocs.org/2014/na/talks/#amalia-hawkins-ignorance-is-strength-writing-documentation-by-learning-as-you-go

Link

"Writing documentation by learning as you go."

We usually think about ignorance as the enemy, something that we want to overcome using documentation. But it's when we have that struggle ourselves, we're in a good position to understand what users need to know.

Hawkins graduated in May, and started working in August. In college, she was the lead TA for an Intro to CS course, and worked on hiring approximately 30 TAs for the course. What she discovered was that the best TAs weren't always they students who effortlessly got A's; the ones that had to work hard to get their B often understood the struggles of incoming students better.

When Hawkins started working at Mongo in August, she was enthusiastic, ignorant, and not much else, and there was > 1MM lines of code to understand. A lack of internal documentation makes scaling the team more difficult.

So after she figured out the problem in front of her (via a conversation with the CTO), she wanted to share what she'd learned. She talked to the documentation head, who suggested: write anything, even if it's wrong. Because even if it's wrong, people now have something to respond to and critique.

And people responded positively: "Wow, I didn't know about that, that's helpful." "I'm so glad someone wrote about this. Someone should also write about X, Y, and Z." So you need to find your allies, and help spread the work around.

1.1.6 Did It In Minutes: The Art of Documenting Meeting Notes

Authors Mo Nishiyama

Time 11:40 - 12:00

Session http://docs.writethedocs.org/2014/na/talks/#mo-nishiyama-did-it-in-minutes-the-art-of-documenting-meeting-notes

Link

See also: "Did It In a Minute", Hall/Oates

Release notes and documentation are like the Corvette that everyone wants to write/read. But meeting notes are the garbage truck: plebian, under appreciated, and mind numbing. Meetings can be toxic and awful places to be.

When Mo was assigned to talk meeting minutes, he realized that he'd been viewing meeting minutes as regurgitation, when they're really about curation.

1. Understand your audience: who's at the meeting, but also who's *not* at the meeting? You're not just writing for yourself.
2. Once you understand your audience, you can create a shared need. What does the audience commonly need to know.
3. Chronology doens't matter; what happens at the beginning of the meeting doesn't need to go a the top of the document.
4. WTF: Write the Facts. You don't need to record the "healthy discussions" or "heated discourse".
5. Engage the Subject Matter Experts: engage and ask questions before you make assumptions and publish incorrect interpretations.
6. Make the action items clear: Who, What, When need to be defined.
7. Use templates to save time: formatting, organization, etc. Helps save time for the writer.
8. Use easter eggs to combat TL;DR

And there are exceptions, of course.

School board meetings require chronology. Depositions require regurgitation. And some organizations require style guide adherance, etc.

1.1.7 Hacking the English Language

Authors Nina Vyedin

Time 12:00 - 12:20

Session http://docs.writethedocs.org/2014/na/talks/#nina-vyedin-hacking-the-english-language

Link

Or, what can we learn about writing from our programmer friends.

When Vyedin was in college, she did really well on her first paper. And when she went to write the second, she blocked at the blank page. Now she works for Xamarin, and when you create a new app using their tool, you get boilerplate and guidance about what to do next. What if you didn't have to start with a blank page when you're starting a document?

Don't start with a blank page.

Create a template for each type of document you write, and put them in the same place, so you never have to start with a blank page.

Make a spec

While mentoring a new writer through a document, Nina realized that she and the new writer thought the document was answering different questions. So you can write a simple spec for your doc, that keeps everyone involved on the same page.

- What question is this document meant to answer?
- Who is the target audience?
- What's the current state of the documentation on this topic?
- What's the work plan?

Programmers also have design patterns: pre-established architectural patterns for software. Documentarians have that, too, and you can use those to make the work easier.

Writing for Clarity: Name your variables

Most programming texts contain a section on naming variables. Writing needs this, too. There are things like the undefined "if", a lack of ownership (passive voice), vague terms ("view", "component", etc), or ideas whose name changes over time, all of which make your writing less clear. Be specific, and be precise.

Editing

Editing isn't just about correcting spelling or punctuation. Editing is the refactoring of writing. Editing should be about rearranging, restructing, and clarifying what's going on in a document. (Just make sure you're not changing the question.) You don't have to be an expert to be a good editor (although it helps); you just need to be willing to ask questions.

1.2 Afternoon

1.2.1 Lighting Talks

Write Tighter, Revisited

Author Marcia Johnston, @MarciaRJohnston, http://howtowriteeverything.com/

Gave a talk last year, "Write Tighter", and she wanted people to remember one thing: **see be-verbs**. (Not beavers.) "be", "was", "has been", "will be", "will have", "been having". Those are a flag that your writing can be tightened up, made more concise, made more engaging, and clarified.

But why?

It's cheaper! Translations cost $0.25/word/language, so getting rid of "be-verbs" means you pay less. And you *will* be translating if you're remotely successful.

And it makes your writing more readable.

Test the Docs

Author Dirk Myers

Partnering with other startups to use their SDK to develop applications to spec. This lets them effectively pilot the new user experience. What they've found is that the bugs filed aren't at all what they expected. You *know* about prolems in your product that you think users will report, but they don't. They want to know *what's next*. So now they write the docs *first*. They write a doc, show it to customers, and ask what they'd do next, what questions they have, etc. And this is leading to increased customer happiness with documentation and reduced support costs.

Open Source Documentation, The Hard Way

Author Anne Gentle, http://justclickwrite.com/

"I'm a unicorn: an extroverted technical writer."

Half of the Open Stack documentation was written by 3 writers. Six months later, another half was written by 7 writers. That was a lot of work. But there were 130 documentation contributors overall. And when there are 910 overall Open Stack contributors, you have to wonder: am I getting the right contributors?

Ran a book sprint to write a 200 page book in a week. Flew in contributors who'd been operating Open Stack at scale for six months or more, and wrote *the guide* to Open Stack. And then O'Reilly asked if they could publish it (now 300 pages), since they were having trouble finding qualified authors. And now they've produced an Open Stack security guide, as well.

Applying developer techniques to documentation.

Write, Measure, Repeat

Author Dan Stevens

New to Atlassian, working on BitBucket. And they're trying to change how they think about information solutions. One thing he doesn't think Atlassian does very well is measure results for documentation. So they asked themselves, "How do we measure success with documentation? And how do we make that repeatable?" Beyond just writing something technically accurate, what are we trying to achieve?

"In the documentation, can we lead them from the software, to the docs, and back, to acheive a specific feature adoption?"

So now they're rewriting tutorials, and measuring how many people are entering documentation, and then acheiving some business goal. [NB: Conversions.]

Measuring conversion lets you talk about documentation as more than just a cost center.

EPUB in a Nutshell

Author Chuck, @chuckdude

[Former O'Reilly editor]

EPUB is a website, packed in a folder, with some XML files. To "do" EPUB, you need to be comfortable with HTML, CSS, and some light XML. What EPUB? It's portable, searchable, bookmarkable; you can *copy and paste*. PDF – the "pretty dead format" – is the bane of a lot of documentarian's experiences. It's great for presentation and dead trees, *not* for sharing.

How do you learn EPUB? Look at the source. Just `unzip` any EPUB file.

[Demo of what an EPUB contains.]

EPUB is pretty simple, and it's incredibly powerful.

EPUB is managed by the IDPF (International Documentation Publishing Forum).

1.2.2 Documenting Domain Specific Knowledge

Authors Alex Gaynor

Time 13:50 - 14:10

Session http://docs.writethedocs.org/2014/na/talks/#alex-gaynor-documenting-domain-specific-knowledge

Link https://speakerdeck.com/alex/documenting-domain-specific-knowledge

Alex got his start writing software working on the Django project, a project with a very strong culture of documentation. So what do users come to the documentation looking for? Some users just want the details. Alex isn't concerned with them: they usually have lots of context to understand what's being written, and they're primarily concerned with the completeness of the documentation. But what about the people who come to the documentation who *don't* know about the request/response cycle, or about how the web works? It's not clear how well the tutorial really works for them (probably not very well).

Alex has been working on a Python cryptography library, and most users just want the documentation to tell them what to do, without gaining actual deep knowledge.

Can you document around a usability problem? Perhaps. What you can't document around is their assumptions: if you don't shock them out of their assumptions, and your software doesn't conform to those assumptions, you'll just create frustration.

So documentation and design are a partnership.

Pick an audience. That's the audience you'll write software for. "People who use my software" is not an audience; an audience is a set of knowledge and background. Sometimes you'll need to write documentation multiple times for multiple audiences. You'll need to partition documentation between those audiences (this can be challenging; one model is the software layer model, which is a good way to separate users from experts).

Being able to write documentation that you can read top to bottom – "straight line documentation" – is incredibly valuable. Writing in a reference style, with lots of links, is difficult to digest. Those links should be optional for most users.

You should also eliminate assumptions you make as you're writing documentation. Remove jargon, wherever possible, and when it's not possible, the explanation should be front and center. Warnings about using your tool the *right* way should **not** be buried in a link. And of course, the design of your code should back up your documentation, and provide sane defaults.

Your documentation should also enable use cases. Most people approach documentation trying to solve a specific problem. Figure out what those are, and write your documentation *for them*.

Composition is great for building software, but it can be problematic for documentation, where users might not know how to connect the dots. *Connecting the dots is your job as an author.* You can test whether you've succeeded by testing your documentation with real users. When they reach for Google, that's a documentation bug.

Nothing of consequence should depend on something the users doesn't know they don't know. Your software (and documentation) can guide them: make recommendations, not defaults, so that users are aware when they're making a choice, and learn more about what they might not know.

A tutorial should give users multiple, easy wins, preferrably with less than 30 minutes of work. Give them points to stop and rest and celebrate their progress.

Your users will never care about your domain the way you do.

1.2.3 Graphical Explanations

Authors Geoffrey Grosenbach

Time 14:10 - 14:30

Session http://docs.writethedocs.org/2014/na/talks/#geoffrey-grosenbach-graphical-explanations

Link

Formerly ran PeepCode, now at PluralSight.

In the late 1980's, the founder of the TED Conferences wrote a book, Inforation Anxiety, about how just getting access to information wasn't enough. People would be willing to pay for curation and filtering. And he actually wrote the book in such a way that you could read it at different levels: you could read the TOC and get something, flip through it and get something more, or really read it and gain deep knowledge.

Can we apply that to documentation? Give people the ability to skim for a graphical representation if they don't have time to read it all, or really dive deep if they have time.

What are some easy ways to do this?

1. Type

   It's the way we communicate, and it's everywhere, so learn to use it well.

2. Color

   (pygments is a great way to handle this formatting, depending on your tool chain.)

3. Icons

   SymbolSet / Symbolicons

4. Explanatory tools

   - Before/After
   - Video
   - Icons to indicate "warnings" or "best paths"
   - Good/Better/Best

CHAPTER 2

Tuesday

2.1 Morning

2.1.1 Putting the (docs) Cart Before the (standards) Horse

Authors Drew Jaynes

Time 9:00 - 9:20

Session http://docs.writethedocs.org/2014/na/talks/#drew-jaynes-putting-the-docs-cart-before-the-standards-horse

Link

WordPress took 10 years of inline documentation in the code and developed a standard from that. Historically there was very little attention given to inline documentation. The phpDoc spec was followed loosely, but the Codex (wiki) was still seen as the main entry point for documentation.

The Codex is a 2300+ page wiki of manually curated content. Before every release, the documentation team would get together and make a list of pages that needed to be updated. As the rate of WordPress releases increased, the pressure on the Codex/Documentation team increased, to the point where it felt unsustainable.

About 8 months ago, the Codex team wanted to document all the hooks in WordPress core. Rather than trying to fit the WordPress hook architecture into an existing standard, like phpDoc or [something else], the team decided to survey existing practices in the code, and developed Hook Docs.

Some background: In June 2013 WordPress Core went through some evolution, and began recognizing teams of contributors. The Docs Team came out of that, which gave the team its own blog, home, etc. The team held an informal summit at the Open Help Conference, where they reviewed the Codex Survey conducted previously, and developed a roadmap.

One of the primary goals of the roadmap was eliminating the Codex. That goal spurred development of Hook Docs.

Eight months in, with the release of WordPress 3.9, all the hooks (2200+) in WordPress Core have been documented. This is an increase from 66k to 99k lines of documentation in the codebase. There were 40 new contributors over 3 releases, and version 1.0 of the code reference is now live, and derived from the source code.

Developing a standard was obviously beneficial to the WordPress, even if the project arrived there slightly later than other projects. (And the Codex will become a giant 301 redirect.) The development of the internal standard – even though it diverges from some other PHP practices – is leading toward longer term consistency for the project.

2.1.2 From docs to engineering and back again

Authors Susan Salituro

Time 9:20 - 9:40

Session http://docs.writethedocs.org/2014/na/talks/#susan-salituro-from-docs-to-engineering-and-back-again

Link

Currently a software infrastructure engineer at Pixar, but over the course of her career she's gone back and forth between engineering and writing. Last year's WTD inspired her; finally, a group of people who care about the things she cares about: communicating with others more effectively.

As a self described introvert, Susan didn't realize until after she'd submitted her talk that she was effectively committing to telling her professional autobiography in front of 300 people. At Pixar they say "the story is king". So what's the story?

Entering college, Susan loved astronomy, and wanted to be a astrophysicist. But eventually left the program, because she wasn't sure she could complete the degree. But she did enjoy math, so she became a math major. And then added an English major to that. And as she considered what career she might pursue, she took the single technical writing course offered by her university. It was more geared toward graduate students trying to write their thesis and dissertations (don't use passive voice, etc), but suddenly she understood that there was a way for her to combine both of her interests.

Over the next eleven years, Susan pursued technical writing, and got a lot out of it. There was a sense of community around technical writing, and a sense of passion shared by the community that was appealing. As a technical writer, there was also a continued sense of learning and beginner's mind: your *job* is to communicate about your product to people who haven't experienced it yet. And finally, there was a lot of flexibility: it's a portable skill that Susan has been able to practice around North America.

But it felt like something was missing: Susan started exploring ways to grow her career. She started seeing technical writers applying their skills to UI design, information architecture, and programming. She visited customer sites with a UI designer, and had the opportunity to see customers actually *using* her help, which was sort of revelatory. "How can I help the UI team make the application so intuitive that my help content isn't needed?" And she started learning about how software is put together, taking a class in Visual Basic so she could prototype interfaces, and a class in Java Doc, which exposed her to the a world she didn't know existed.

While working at Palm Source, a manager took a chance on her and she was able to dive into documenting the API of one of PalmOS's components. After moving on to Pixar, she was working on a documentation project that used DITA, an XML based documentation system. To do this effectively, she had to really expand her skills: XSLT, Python to drive it, Make to automate those. She was a programmer.

"You can't go around writing software without attracting attention."

Susan attracted the attention of the software release engineering team's manager. And this was like drinking from the firehose: there was so much information being thrown at her, she could barely keep up.

And the community was still there, but instead of coming in the form of professional organizations, it came from the company internally. There wasn't as much flexibility as there was with technical writing; you were almost always on call. But it was really difficult.

Susan moved from Pixar to a smaller company, into an information architecture role. This allowed her to combine her technical writing and programming skills. Unfortunately the company went bankrupt, and Susan returned to California, to Pixar. At Pixar she became a Software Infrastructure Engineer in documentation, further blending her techincal writing and software engineering. Her mandate was to find solutions for documentation problems, addressings both tools and process issues. But the focus still shifted to product life cycle, as opposed to writing.

This isn't the end of the road. There's still more to learn – for both Susan and you [in the audience]. Find a community, take a risk, make a choice.

2.1.3 Don't Write Documentation

Authors James Pearson

Time 9:40 - 10:00

Session http://docs.writethedocs.org/2014/na/talks/#james-pearson-don-t-write-documentation

Link http://changedmy.name/talk–dont-write-documentation/

As you may have guessed by the title, James is not a writer. But the scraggly beard and EFF sweatshirt indicate that he *is* a sysadmin [for iFixIt], and as such documentation is just about making sure you don't miss any steps. iFixIt is a wiki-based platform that lets any user write documentation and step-by-step guides to repair anything.

So why *shouldn't* you write documentation? Because good documentation's worst enemy is bad documentation. The programmers who James hangs out with *hate* documentation; they *assume* that it's going to be wrong, so they don't even try to read it. For example, the image processing library they were using to annotate images had many, many bugs. Upgrading fixed some and introduced others. The take away was that that library is crap. But the take away when we see bad *documentation*, is that *all* documentation is bad. We don't treat documentation like software (where we might think this *particular* piece of software is crap): we're numb to documentation.

See also: broken window syndrome. When you see bad documentation, it tells you that lax standards are acceptable for this project or product.

So what sort of documentation is awful?

Autogenerated documentation can be awful.

The Perl "Camel" book is a joy to read, and it instructs us on the Three Virtues of a Programmer.

- Laziness
- Impatience
- Hubris

Laziness is the one that causes pain. Tools like Javadoc are often abused: just because your tool can extract documentation doesn't mean that the documentation is going to appropriately communicate content to other users.

Poka-Yoke – "mistake proofing".

A Toyota engineer realized that people will always make mistakes (because we're human), but mistakes don't have to become defects. Poka-yoke refers to designing a tool or product in a way that prevents mistakes from becoming defects. (See http://pokayoke.wikispaces.com/ for examples.) The solution to many defects is not documentation, but changing the product or software.

2.1.4 Make Music Not Noise

Authors Christopher Kelleher

Time 10:00 - 10:20

Session http://docs.writethedocs.org/2014/na/talks/#christopher-kelleher-make-music-not-noise

Link

Making sound is not the same as making sense.

Information is not the same as communication.

And the standards of music can guide us torward communication.

So what is noise? We usually think of it as "what I don't like to hear". But more formally, it's "sound without structure". The structure is patterns and frameworks, and the search for structure appears to be hard-wired into our brains. Listening to unfamiliar music in an MRI, we see both the most ancient and most modern parts of the brain lighting up as the subject begins identifying patterns and structure. Even infants respond to pitch and harmony: sound with structure.

And while we reject “noise” as “not songs” almost instinctively, we don’t have the same hard-wired detection for poor documentation.

Most noise is inadvertant, but sometimes it’s intentional. [Shows the iTunes TOS.] It’s tempting to think that the TOS is a failed document, because it’s so long and inscrutable. But it’s not, because it’s not an attempt to communicate: its goal is to beat you into submission an acquiesence.

By neglect, we have the expectation that documentation is supposed to hurt. But we can do better. Music-like satisfaction is achievable and desirable. And we can get there.

2.1.5 Instrumentation as Living Documentation: Teaching Humans About Complex Systems

Authors Brian Troutwine

Time 10:40 - 11:20

Session

Link

Troutwine is a software engineer: he writes software that talks to other software, not necessarily end users. These are real time, distributed systems, and he currently writes this software for Ad Roll. That might seem like an odd fit, but advertising is changing from large accounts that spend $10,000 per month on print advertising to small, real-time bidding on ad slots (ie, $10/month on advertising for fork bracelets on Etsy). The problem is a low latency, firm real-time system, with non-smooth traffic patterns. This is considered a Complex System.

Complex systems have non-linear feedback that are tightly coupled to external systems. They’re difficult to model and understand. They often try to solve “wicked problems” [as described by C. West Churchman]. Complex systems fail in catastrophic ways, and when they do, they often create problems worse than those they set out to solve.

It turns out that just because we make something doesn’t mean we understand it (see Carlos Bueno, “Mature Optimization Handbook”). The key challenge is actually maintaining and increasing our understanding of the problem *and* the solution. So we write documentation. (Which is troublesome, since communicating about these complex systems are difficult to communicate about.)

Miscommunications are accidents in the making: the Challenger disaster happened because there was miscommunication about the temperature constraints on the O ring. If you don’t know how a system *should* behave, you can’t say how it shouldn’t or isn’t behaving. And the documentation doesn’t keep up with with the system.

Eric Schlosser, “Command and Control”

So if complex systems are so difficult to build and maintain and document, what can we do?

Instrumentation reflects the reality of the system *as it is*, not as we believe or expect it to be. Exploration guided by instrumentation, done honestly, guides us to a better understanding of the system.

[Instrumentation case studies.]

And it’s possible to have too little information from instrumentation, or even too much (3 Mile Island example). Too much information hinders interpretation. Instrumentation isn’t a panacea. It’s also software itself, so *it’s* susceptible to bugs, as well!

2.1.6 We Strongly Recommend You Write Best Practices

Authors Lauren Rother

Time 11:20 - 11:40

Session http://docs.writethedocs.org/2014/na/talks/#lauren-rother-we-strongly-recommend-you-write-best-practices

Link

Works on Puppet Forge at Puppet Labs; Forge is a tool for hosting user contributed modules and discovery. As the Forge has grown, users have been asking for Puppet endorsed best practices and practices. Documentation at Puppet Labs is embedded with engineering, so the content is developed jointly with engineers who are writing Puppet modules.

Puppet users have different needs, workflows, and most importantly, different levels of experiences. The experiment was the "Beginner's Guide to Modules", targeting novice users. They've chosen to target different levels of user experience as the primary differentiator.

In the first version, they dumped everything they thought users might ask or wnat to know into a document, then organized it into what felt like logical sections using a textbook-like approach. The problem was that it was overwhelming. The large paragraphs of text felt like they were difficult to enter into and really understand. The solution was to provide Orientation: introductory sentences that describe what you're about to read, and sign-posts along the way to orient yourself. ("Hrm, I was supposed to know how to scope a module after reading this section; did I actually learn that?").

They'd initially rejected this as overly-verbose and excessive text, but what they discovered was that it makes the document more accessible. This is true for novices – who get some context as they read – as well as more experienced users – who can use the signposts as a way to jump over content they're already familiar with. [NB: This means it's really important that the signposts are *accurate*.]

But even with smaller paragraphs, introductory sentences, and sub-headings, it was difficult to locate distinct items. It was better for the intial read, but still challenging when you returned to the document and wanted to pick up where you left off. It was readable, but not usable.

Thinking about the problem, Lauren realized that they were writing a treatise, not a guidebook: something you read as a challenge, not something that you return to again and again.

Best practices are often really a step by step guide, just like NKOTB told us.

The final iteration has a table of contents, headings, introductory sentences, steps (which makes it easy to write the TOC), and *examples*. Interestingly, the process of putting the guide into steps made it easier to add more examples, which give context and background that words can not.

There's still room for improvement: the beginner's guide works well and provides a good overview, but it's very high level. It glosses over questions that will come up very quickly as users get up to speed on Puppet modules.

2.1.7 Wabi-Sabi Writing

Authors Scot Marvin

Time 11:40 - 12:00

Session http://docs.writethedocs.org/2014/na/talks/#scot-marvin-wabi-sabi-writing

Link

So what is Wabi-Sabi? It turns out that the only people really talking about it are westeners. It comes from Japanese Zen Buddhism, where there's an appreciation for syntactic ambiguity, something we don't have much of in the west.

The tenets go something like this:

- Beauty in the impermanent
- Beauty in the imperfect
- Beauty in the incomplete

Now as strict Helenists, this idea of imperanence and imperfection doesn't always sit well. But when have we written software that was even *complete*, let along *perfect*?

Wabi-sabi is sometimes described as rustic or primitive, but that doesn't mean it's the same as Luddism.

And just because it embraces imperfection doesn't mean that it's complacent. You still iterate, you still work on your product, but you don't expect that an iteration will be perfect.

Agile is also about incremental betterment.

1. Less Faulkner, more Hemingway; or, clear writing.

   Faulkner and Hemingway represent two extremes of 20th century American writing. You can lose yourself in Faulkner's sentences, but it makes for a poor paradigm for technical writing and documentation. Hemingway is the king of the short sentence, and that's something to aim for in technical writing.

2. Less Coltrane, more Davis; or, tranquil writing.

   Coltrane was known for his controlled chaos: he put so many notes into a measure, it was sometimes known as "sheets of sound". Davis, on the other hand, was the king of calm and cool, demonstrating economical restraint in his compositions. When readers come to documentation, it's unfortunately usually as a last resort. They need calm and cool, not controlled chaos.

3. Less Versailles, more Ryoan-ji; or, simple writing

   Marvin talks about loving things that are like the Gardens of Versailles: magnificent, elaborate, and intricate. But most of our readers usually want something closer to a zen sand garden.

4. "Done is beautiful."

   We can't *complete* anything, but we can get something done. And done is always beautiful.

Embrace the crack and crevices of your work.

2.1.8 Strategies to fight documentation inertia

Authors Britta Gufstafson

Time 12:00 - 12:20

Session http://docs.writethedocs.org/2014/na/talks/#britta-gustafson-strategies-to-fight-documentation-inertia

Link

Worked on reviving volunteer documentation wiki with a little social engineering and friendliness. The wiki for years had an outdated tagline, and didn't link to the great content that was actually there. Now it contains explicit call-outs to getting started, beginner information, and entry points that people coming to the community may be interested in.

Britta joined Cydia as a community manager, and tried to push engineers towards updating the wiki. After getting involved in Open Hatch, she realized that she actually *did* have something to contribute to the wiki and to the documentation.