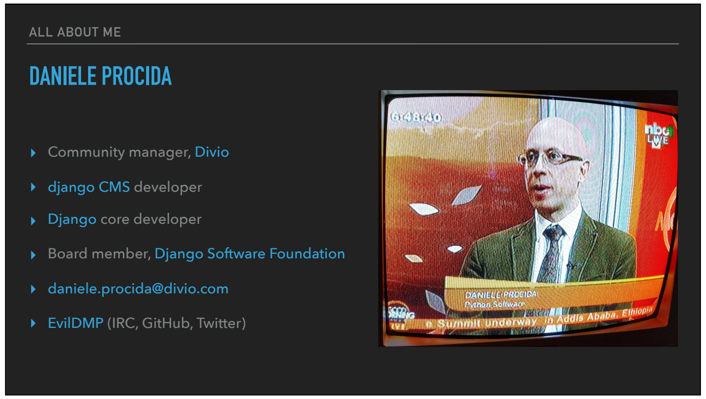
DANIELE PROCIDA

DOCUMENTATION-DRIVEN DEVELOPMENT



ALL ABOUT ME

DANIELE PROCIDA

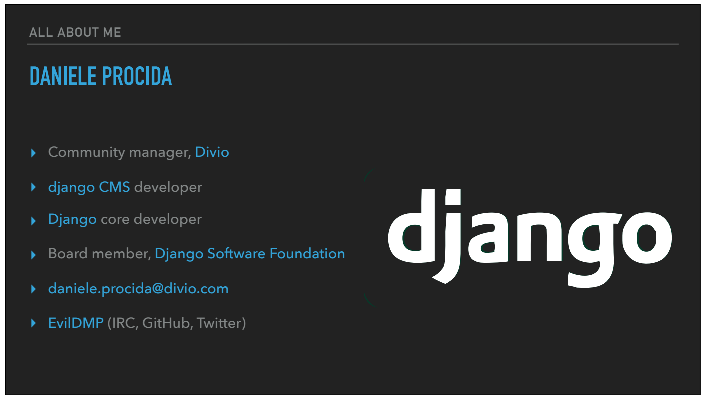
- Community manager, Divio
- django CMS developer
- Django core developer
- Board member, Django Software Foundation
- daniele.procida@divio.com
- EvilDMP (IRC, GitHub, Twitter)

I work for Divio, a Swiss-based Django software and services company.

Part of my job description is *documentation manager*, and I think consider myself very fortunate to be working at a company that considers documentation important enough that someone should be a documentation manager - I don't think it's usual, especially in a company of this size.

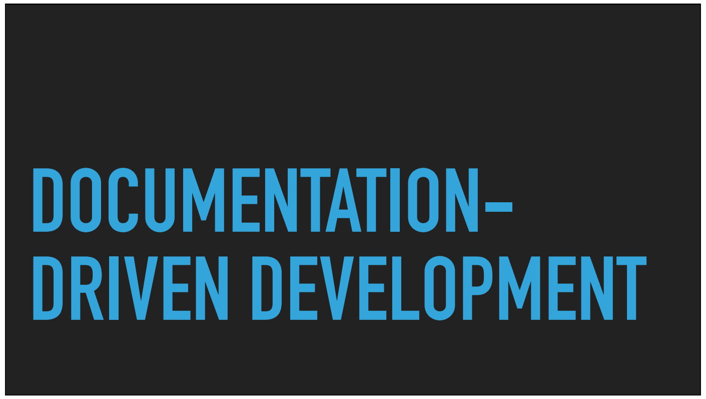
DANIELE PROCIDA

- Community manager, Divio
- django CMS developer
- Django core developer
- Board member, Django Software Foundation
- daniele.procida@divio.com
- EvilDMP (IRC, GitHub, Twitter)



DANIELE PROCIDA

- Community manager, Divio
- django CMS developer
- Django core developer
- Board member, Django Software Foundation
- daniele.procida@divio.com
- EvilDMP (IRC, GitHub, Twitter)

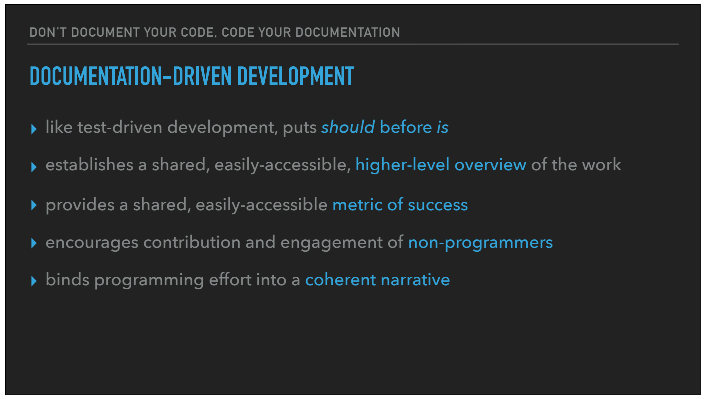


DOCUMENTATION-DRIVEN DEVELOPMENT

Some people take quite seriously the idea that you should write your documentation first, and that your code should follow.

It's not nearly as commonly discussed or practised as **test-driven development**.

I've never heard anyone actually speak about it, or met anyone who says they do it.



DOCUMENTATION-DRIVEN DEVELOPMENT

- like test-driven development, puts *should* before *is*
- establishes a shared, easily-accessible, higher-level overview of the work
- provides a shared, easily-accessible metric of success
- encourages contribution and engagement of non-programmers
- binds programming effort into a coherent narrative

I don't want to spend too much time here, but documentation-driven development reverses the typical priority of code and documentation. You start with the documentation instead of your code, and instead of documenting your code, you code your documentation.
And I don't actually know very much more about it, though I am sure it is a valuable development discipline that more people should adopt.

In fact, what I want to talk about is not this discipline, but some **other** senses in which **documentation drives development**.

DOCUMENTATION-DRIVEN DEVELOPMENT

LESSONS FROM THE DJANGO PROJECT

So let’s have a look at Django, and consider what documentation has meant for its development.

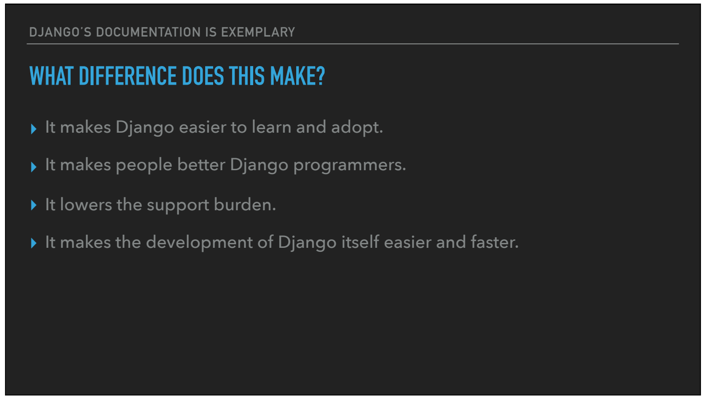
DJANGO'S DOCUMENTATION IS EXEMPLARY

I think the first thing we should say, not because it's a new observation but because everyone seems to agree about it, is that *Django's documentation is exemplary*.

I've not come across any similar project with better documentation - perhaps my experience is limited, but no-one else seems to have discovered better documentation either.

WHAT'S SO GOOD ABOUT IT?

- It's structured properly (tutorials, how-to, reference, topics).
- Within that structure, it's clear and consistent.
- It covers just about everything.
- It's held to the highest standards.
- It exemplifies important values (clarity, courtesy, friendliness).
- Documentation in Django is a process, not just a product.

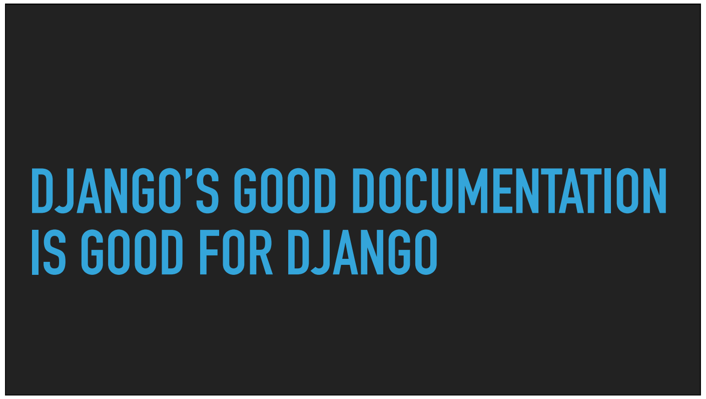


DJANGO'S DOCUMENTATION IS EXEMPLARY

WHAT DIFFERENCE DOES THIS MAKE?

- It makes Django easier to learn and adopt.
- It makes people better Django programmers.
- It lowers the support burden.
- It makes the development of Django itself easier and faster.

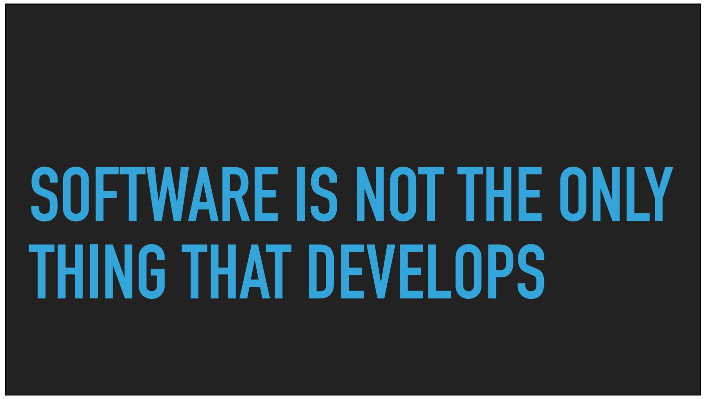
Some of the effects are obvious.



DJANGO'S GOOD DOCUMENTATION IS GOOD FOR DJANGO

So Django's documentation, without any doubt, has been good for Django's development.

And here we come to the main point of my talk:



SOFTWARE IS NOT THE ONLY THING THAT DEVELOPS

Software is not the only thing that develops and grows and improves.

Programmers, and **communities of programmers**, **also** grow and develop and improve, and **they** are what makes the software develop.

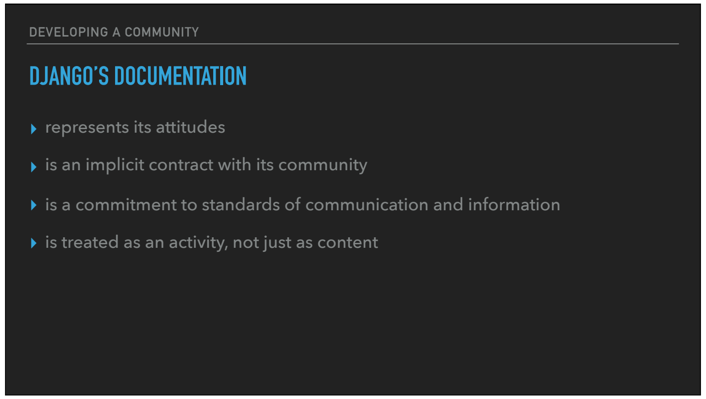
WHAT DOES DOCUMENTATION MEAN FOR THE DEVELOPMENT OF COMMUNITIES AND PROGRAMMERS?

So the question I am particularly interested in is **what does documentation mean for the development of communities and programmers?**

DEVELOPING A COMMUNITY

Django’s community, like its software, is stable, mature and dependable. It contains few unpleasant surprises. It’s active, engaged and remarkably united. It has difficulties, but not crises or lingering ills.

I think that one of the glues that has bound it together is in fact Django’s documentation.


DEVELOPING A COMMUNITY

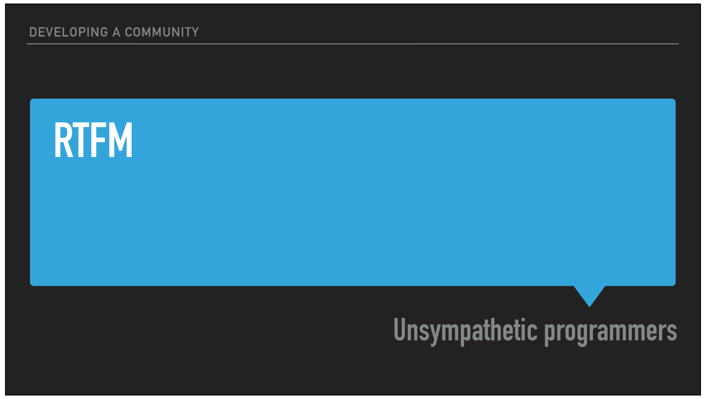
DJANGO'S DOCUMENTATION

- represents its attitudes
- is an implicit contract with its community
- is a commitment to standards of communication and information
- is treated as an activity, not just as content

I think that when it comes to the development of its community, Django's documentation does four very important things.

I said earlier that Django's documentation **represents its attitudes**, but it's even stronger than that; the care that the documentation takes is an **implicit contract** with its community, and forms a **commitment to standards** of communication and information.

Django treats its documentation as an **activity**, not just as product, as content. This is the deepest of these points, and I will start with it.



DEVELOPING A COMMUNITY

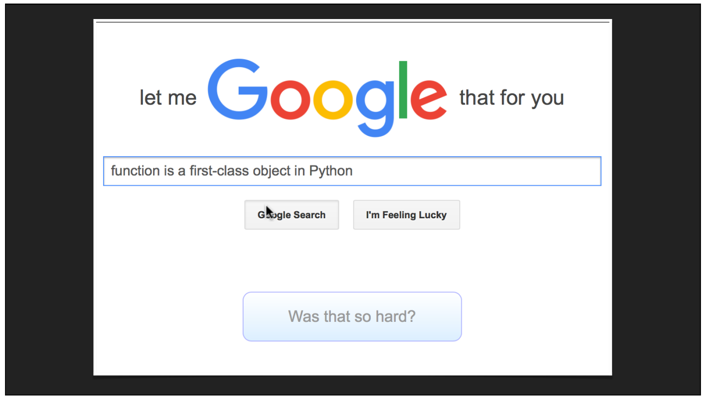
RTFM

Unsympathetic programmers

Have you ever hesitated to ask for help with a programming question, for example on IRC, because you felt that the answer was already out there if only you knew how to find it, or perhaps, how to understand it, but that you might be invited, dismissively, to Read the Fucking Manual by someone who thought you were just being lazy or stupid?

You might be right to hesitate; people who already know things can be remarkably forgetful about how they learned them, or what it was like not to know them.

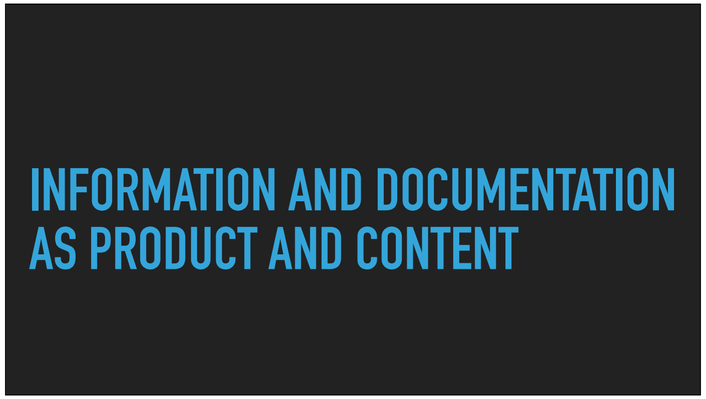
People are not always sympathetic and friendly to people who don't yet know things that they do.


let me Google that for you
function is a first-class object in Python
Google Search
I'm Feeling Lucky
Was that so hard?

Or perhaps you've asked a question and someone has replied with a link to the sarcastic *let me Google that for you* website.

It's a horrible website. I dislike it intensely, and repudiate what it stands for.

It stands for putting down people who are asking for help in understanding something, and telling them that the reason they don't understand what they want to know is that they are too stupid or lazy to learn it.

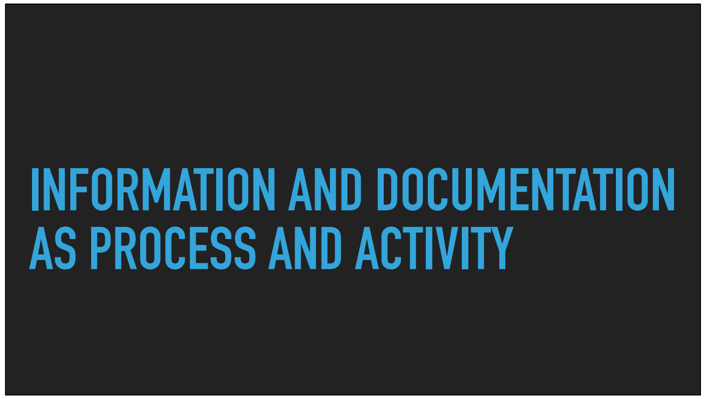


INFORMATION AND DOCUMENTATION AS PRODUCT AND CONTENT

Insofar as information and documentation are considered *content*, it's possible to think and respond like in this kind of way.

The content is out there, and it's freely available to anyone with an Internet connection, so go ahead, help yourself to it and if you don't, **that's your problem**. So, read the fucking manual, or use a search engine.

Other people managed.

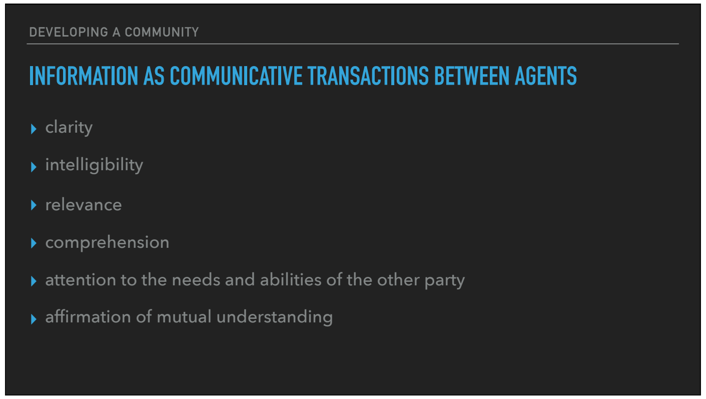


INFORMATION AND DOCUMENTATION AS PROCESS AND ACTIVITY

But if you understand that **information is the process of informing**, an activity, and that **documentation is the process of documenting**, then this response is less possible.

And in fact, we see it far less in Django than we do in other places.

Our IRC channel and email lists are friendly places, where the experts of the community regard information and documentation as **activities** they engage in, not as **stuff** people should have read instead of wasting other people's time.



DEVELOPING A COMMUNITY

INFORMATION AS COMMUNICATIVE TRANSACTIONS BETWEEN AGENTS

- clarity
- intelligibility
- relevance
- comprehension
- attention to the needs and abilities of the other party
- affirmation of mutual understanding

Information, in this model, is regarded as **communicative transactions** between agents.

Information, in this model, demands that we respect values of clarity, intelligibility, relevance, comprehension, attention to the needs and abilities of the other party, affirmation of mutual understanding and so on.

When you’re doing that, you can’t pretend you are “informing” people by telling them to read the fucking manual or sarcastically Googling it for them.

GOOD DOCUMENTATION SHOWS RESPECT

In software, good documentation, and a default position that if someone doesn't understand it then the problem lies in the documentation rather than the person who struggled to understand, becomes a sign of respect, an expression of those values.

Django's documentation sets standards, expectations and the tone for communication, especially for communication with less expert users of it.

It's an assertion of values that are subsequently reflected across the community.

What all this means, I think, is that in this community, in some contexts at least, we do think of information as the activity of informing, something we do, rather than as a collection of content, and that this idea of information has had real, meaningful, beneficial consequences for people who use Python and Django.

DJANGO'S DOCUMENTATION INFORMS ITS COMMUNITY

I said earlier that Django's documentation has been good for Django, but really, I think it has been much more than merely good.

To inform is literally *to shape* something, *to press a shape* into it.

Django's documentation has literally informed, shaped its community.

It has determined how the community has developed, what sort of thing has developed into, and it's one of the things that continues to drive its development.

And that is the kind of development I am speaking of, in **documentation-driven development**.

LESSONS FROM THE DJANGO PROJECT

TASTE THE DIFFERENCE

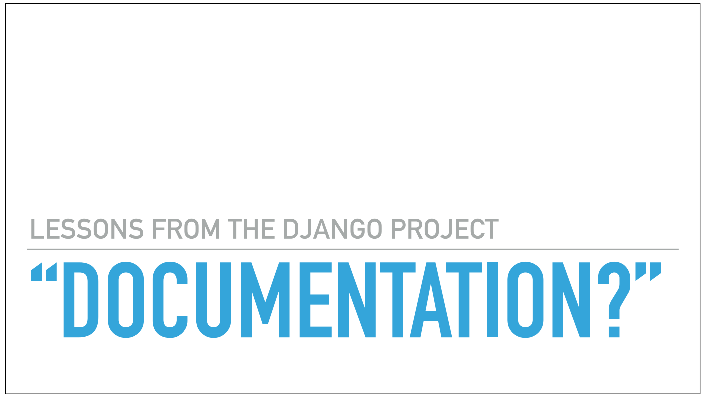
I want to take a brief digression here.

The attitude towards documentation in Django has made a tangible difference that you can experience for yourself in ordinary, everyday ways - you don't have to take a high-level view of the ecosystem or community to see it.

It's brought to me quite forcefully sometimes when I speak to people outside the cosy world of Django. I have an interesting experience.

If I tell a software developer that I'm a member of the Django core development team, then - if they know what Django is - they seem suitably impressed. Then they ask me what I work on, and the answer is: *documentation, mainly*. Or I tell them that part of my job description is *Documentation manager*.

I'm quite proud to be a documentation manager. I'm very pleased that Divio is enlightened enough to pay someone a salary to be a documentation manager, that the company considers it worth spending money on. I feel lucky to be working for a company that has this kind of attitude.


LESSONS FROM THE DJANGO PROJECT

“DOCUMENTATION?”

But, other programmers from outside the Python community can find it hard to hide their sudden disappointment, even embarrassment or a species of bewildered incomprehension - *… documentation?*

It’s as if they thought they were being introduced to Superman and now it turns out it’s just Clark Kent. Sometimes, I get the impression that that one of the possibilities crossing their mind is that I am telling some sort of joke in quite bad taste.

I do believe that once or twice I have even seen the thought flash across someone’s mind: “But - *documentation*? Isn’t that - a *woman’s* job?”

It’s as though I were admitting to some unmanly personal habit, like crying and whining a lot, or writing a function when I should have created a class, or using the wrong kind of workflow on Git, or doing something else that “real programmers” don’t do.

But when I say to a Python or Django user that my main role is contributing to documentation, typically the reaction I get is *Oh, documentation! Cool!*

NINJAS AND ROCKSTARS

So, what sort of attitudes do you expect people in programming communities to have towards things like documentation, when so many of them think they should be rock stars or ninjas? If the ideal of the programmer is a ninja or a rockstar, what kind of space is left for activities such as documentation?

Where did this come from? Why do companies think they might benefit in some way from employing rockstars or ninjas?

Ninjas are famous mainly for setting fire to buildings in the dead of night, which is an interesting metaphor upon which to base a company’s recruitment strategy.

As for the notion of employing rockstars - the mind boggles.

Really?

What are the defining characteristics of a rock star? What comes to mind are things like excessive behaviour and unreasonable demands.

The whole thing seems unbelievably immature.

You'd think that maybe airline pilots would be a more appropriate metaphor.

Or surgeons, or chefs, or anything at all, in which being disciplined, thoughtful and highly-skilled and being able to collaborate well with other people to achieve good outcomes, would be better metaphors for excellent programmers.

NINJAS AND ROCKSTARS

But no: ninjas and rockstars. Literally, arsonists and arseholes.

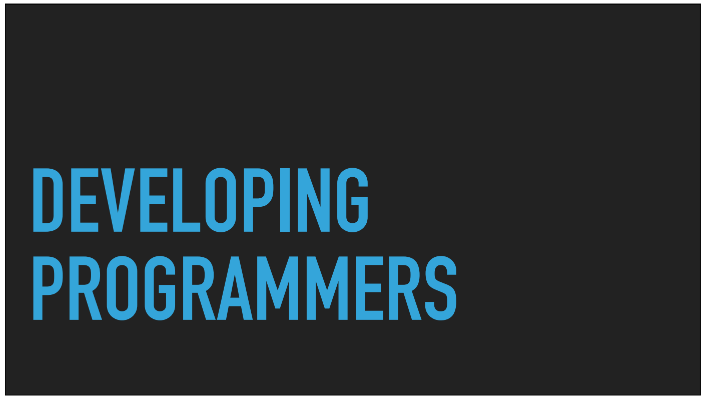
This sort of thing can be dismissed as childish and immature, but it both represents a way people do in fact think about software development, and affects how they think about it.

It makes the companies and the projects where such attitudes are allowed to prevail worse places to be, and harms them: **it has negative effects on their development**.

DJANGO'S DOCUMENTATION INFORMS ITS COMMUNITY

And genuinely, I think that Django's documentation has been part of what has helped Django avoid this.

So it has informed, shaped, the Django community. It has made the community a better place to be. It really has helped develop the Django community.



DEVELOPING PROGRAMMERS

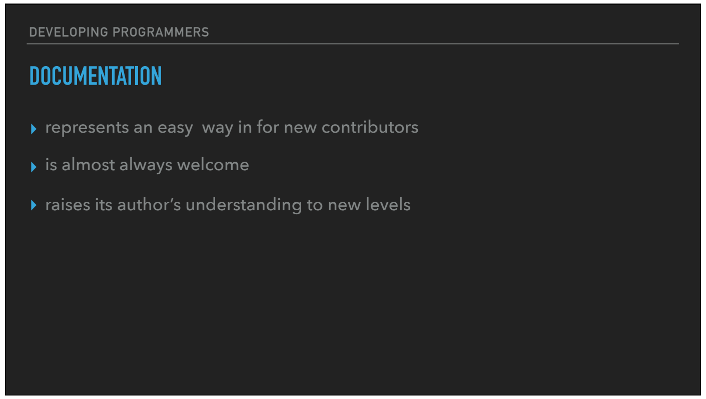
Documentation has implications for programmers in similar ways.

Developers develop, which is to say, their programming skills develop, get better.

The question for many programmers is: how do I develop, become a better programmer?

It’s a true but fairly uninteresting observation that good Django documentation helps Django programmers write better code.

In fact documentation has wider implications than that.


DEVELOPING PROGRAMMERS

DOCUMENTATION

- represents an easy way in for new contributors
- is almost always welcome
- raises its author's understanding to new levels

Documentation is an excellent way for **newcomers** to start contributing to open-source software. You don't need to be an expert in something to be able to identify something unclear or lacking in its documentation and suggest a way to improve it.

Writing documentation represents easy progress to the next step from being a user to being an active contributor of a system.

Documentation is **almost always welcome**; in fact in most projects it's not so much welcome as desperately needed. It's far easier to get new documentation accepted than a new feature.

And **explaining** something to someone else is just about the best possible way to explain it to oneself, to learn and understand it.

So if part of a programmer's development is to contribute and understand more, documentation is an ideal way to do it.

The Django project understands all this.

DJANGO'S DOCUMENTATION GUIDES NEW CONTRIBUTIONS

Django's documentation structure guides and encourages new contributions and contributors.

In Django, the clarity of the documentation's structure makes it almost obvious how and what to write for a particular section, just like well-written code does.

This works just as well for large contributions, such as an entire new section in the tutorial, or for tiny ones, such as an aspect of some key function that deserves more explanation.

CONTRIBUTIONS TO DJANGO'S DOCUMENTATION ARE TAKEN SERIOUSLY AND HELD TO THE HIGHEST STANDARDS

Contributions to Django's documentation are taken seriously and held to the highest standards.

This means that contributors and contributions to documentation receive as much support and engagement as those to code. In many other projects, documentation is so desperately needed that almost any contribution will be accepted just because it's documentation and someone cared enough to write some.

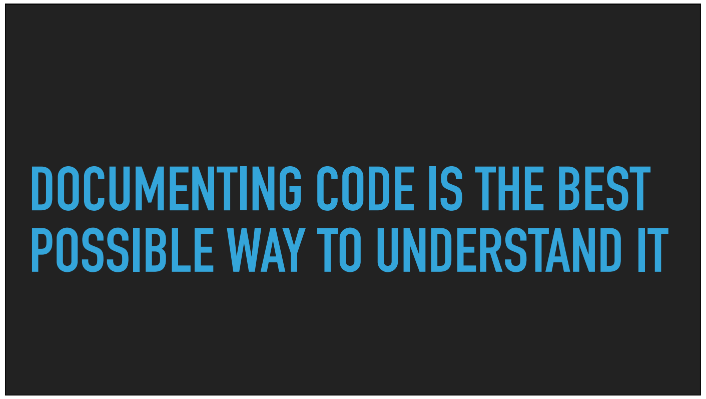
In Django documentation goes through the same wringer that code goes through, and a pull request will be returned time and again until it's perfect.

In one sense this can be tiresome, when it's your pull request that needs to be improved before it's accepted. But in another sense, it's a message that what you're contributing is as important and as valuable as code, and therefore that you, contributing documentation, are as important and valuable as the contributors of code.

CONTRIBUTIONS TO DJANGO'S DOCUMENTATION ARE VALUED

Contributions - and the contributors who make them - to Django's documentation are valued and recognised.

The members of Django's core team are people who have made substantial contributions to Django: mine have all been contributions to its documentation.



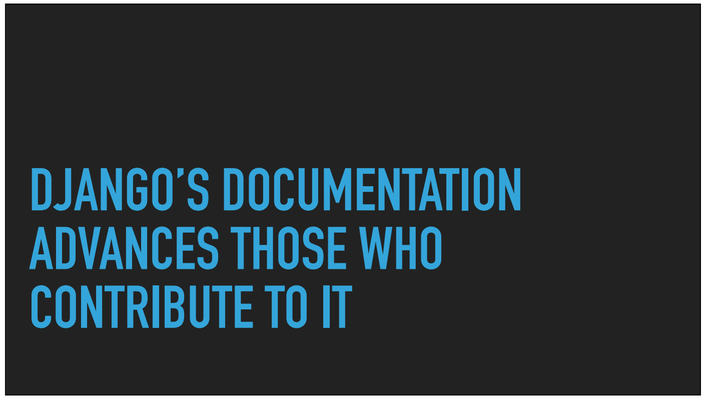
DOCUMENTING CODE IS THE BEST POSSIBLE WAY TO UNDERSTAND IT

These three things - that the documentation guides contributors to it, that documentation is taken seriously, that contributions and contributors to it are valued highly - is something that Django gets right in important ways.

They're especially important because of a fact, one that many people *say* they recognise, but few seem to act as though they do.

And that's this, that **documenting code is the best possible way to understand it**.

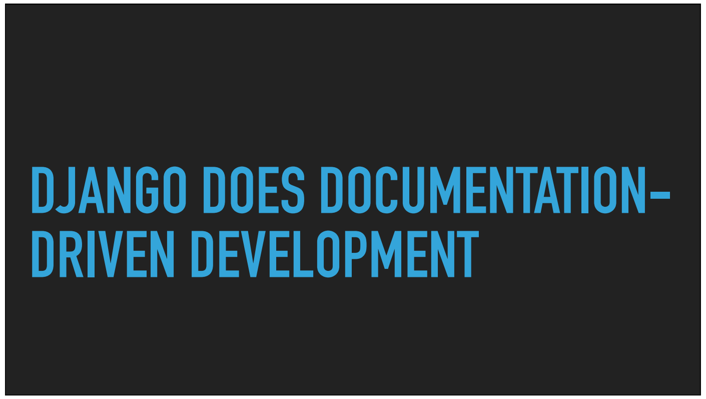
Documenting code will make you a programmer who understands more and understands more deeply - and Django encourages developers to write documentation.



DJANGO'S DOCUMENTATION ADVANCES THOSE WHO CONTRIBUTE TO IT

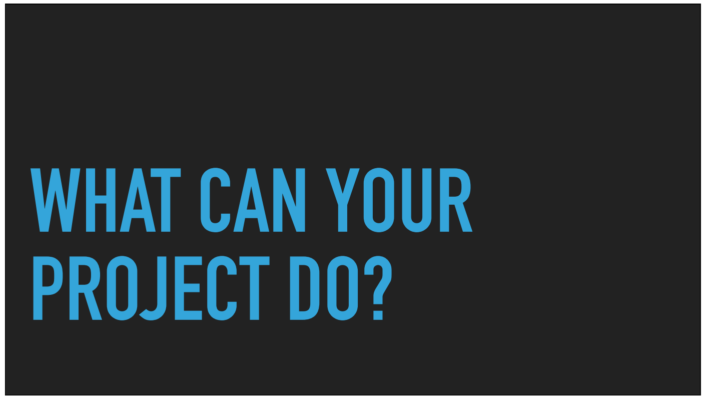
All of this means that Django not only gets more and better contributions to its documentation than other projects do, it also advances the skills, confidence and usefulness of the developers who contribute it.

If you want to learn how to contribute to open source software in Python, there's hardly a better way to start than by doing some work on Django's documentation.



DJANGO DOES DOCUMENTATION-DRIVEN DEVELOPMENT

So I this is what I mean when I say that Django does documentation-driven development: I mean that through its documentation it develops, advances, improves both its **community** and its **developers**.



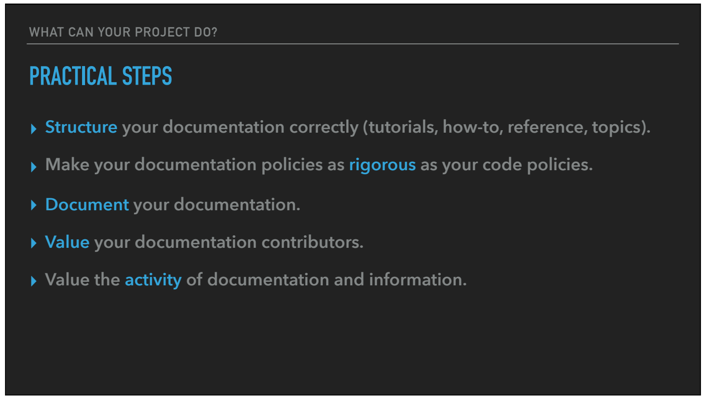
WHAT CAN YOUR PROJECT DO?

That's the lesson from Django.

What can your project do to reap some of the same rewards from its documentation?

I don't think it's something that can be accomplished overnight. Much of this is to do with attitudes, and attitudes are notoriously hard things to change.

On the other hand, some of the actual steps you can take are easy, and I think that if you keep taking them in the right direction, eventually attitudes will follow.


WHAT CAN YOUR PROJECT DO?

PRACTICAL STEPS

- Structure your documentation correctly (tutorials, how-to, reference, topics).
- Make your documentation policies as rigorous as your code policies.
- Document your documentation.
- Value your documentation contributors.
- Value the activity of documentation and information.

Structure your documentation correctly, into tutorials, how-to, reference, topics. See the first page of the Django documentation for a description of them. https://docs.djangoproject.com/en/1.9/#how-the-documentation-is-organized

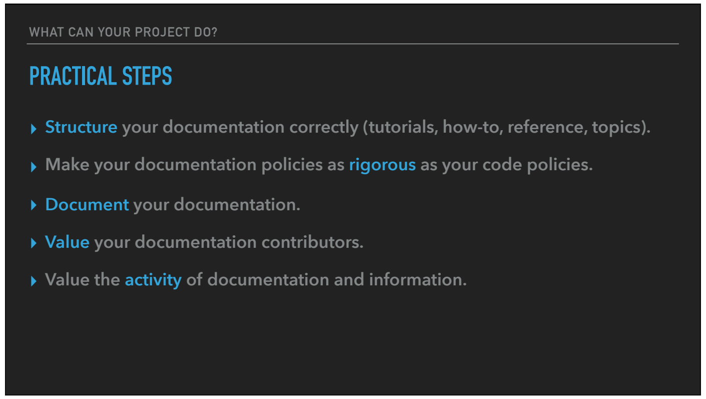
In brief:

Tutorials need to lead users step-by-step to success, to give them confidence and an appetite for more. They should do the minimum of explanation; explanation comes later. At this stage, the user will learn by doing things and using the system. A tutorial should be like teaching a child how to prepare a simple dish, starting with things like how to wash your hands and use a knife safely. And at the end, the child should be able to enjoy the food.

How-to documentation is like a recipe. You can assume some competence and basic understanding. It’s still step-by-step, but it’s more advanced and practically oriented than aimed at pedagogy.

Reference material describes the machinery of the system. It explains its inner workings, or the use of a particular component of it, rather than how to achieve some particular end with it - it’s like a technical description of the different kinds of pasta that go with different sauces, or a list of cooking temperatures.

Topic documentation is discursive; it takes a step back and looks at a topic from a higher level. It can describe and explain general principles, design decisions and so on. It’s useful for putting things into context, like a book about food and eating.


WHAT CAN YOUR PROJECT DO?

PRACTICAL STEPS

- Structure your documentation correctly (tutorials, how-to, reference, topics).
- Make your documentation policies as rigorous as your code policies.
- Document your documentation.
- Value your documentation contributors.
- Value the activity of documentation and information.

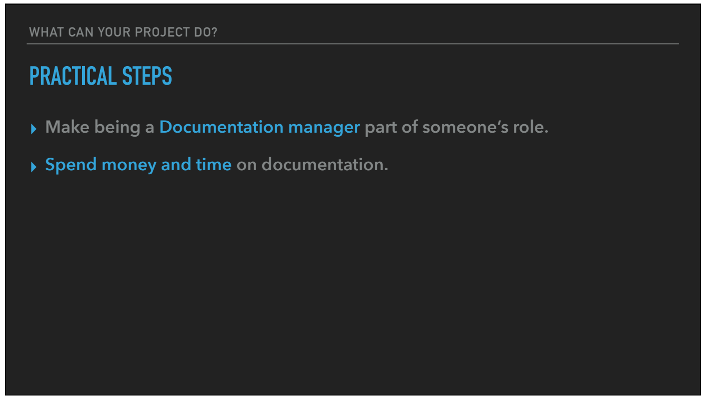
Make your documentation policies as rigorous as your code policies. Don't be afraid to bat documentation back at its contributors: take them, and their contributions, seriously. You wouldn't accept substandard code from them, or code without review; don't accept their documentation simply because it's documentation. They'll appreciate being taken seriously.

Substandard code can harm your applications; substandard documentation is detrimental to your community.

Document your documentation. Make sure it's clear what those policies are, and what you want from each section of the documentation.

Value your documentation contributors. Recognise them, publicly.

Value the activity of documentation and of information. Set aside time for it.


WHAT CAN YOUR PROJECT DO?

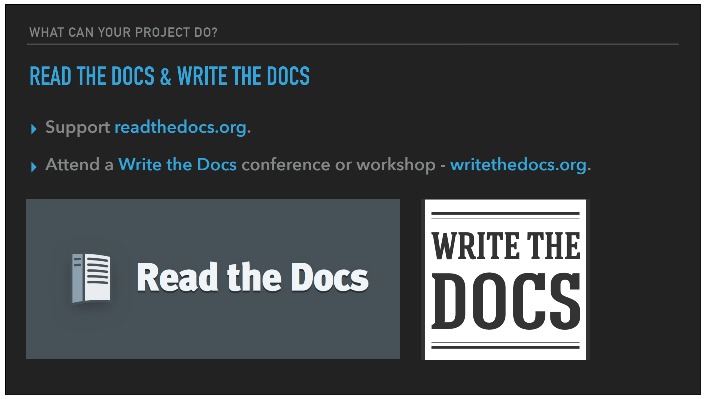
PRACTICAL STEPS

- Make being a Documentation manager part of someone's role.
- Spend money and time on documentation.

If you, or your project or your organisation or your company are serious about this, there are some things you can do - real commitments that you can make.

For example, make being a *Documentation manager* part of someone's role. Pay someone to have that responsibility. Spend money and time on documentation.

All of these will help you on the way to achieving the things on the previous page, and above all will send a message about what documentation means in the company or project.



WHAT CAN YOUR PROJECT DO?

READ THE DOCS & WRITE THE DOCS

- Support readthedocs.org.
- Attend a Write the Docs conference or workshop - writethedocs.org.

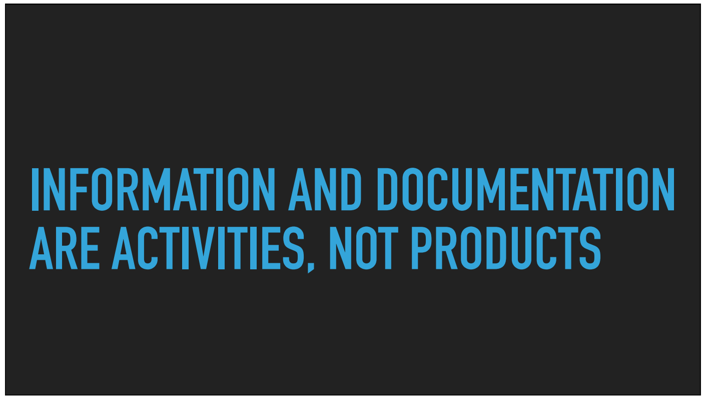
Right here in the Python community, we have one of the most important and valuable resources anywhere: Read the Docs.

Read the Docs is free, it works brilliantly, and it’s cruelly underfunded.

Support Read the Docs. Sponsor it.

Attend a Write the Docs writethedocs.org conference or meetup - writethedocs.org. They can transform the way you approach your documentation, and in turn, the way you develop your software.

Read the Docs and Write the Docs are part of the Python world, and emerged from it - Python’s respect for documentation is one of the reasons for this.



INFORMATION AND DOCUMENTATION ARE ACTIVITIES, NOT PRODUCTS

If you're persuaded by this analysis of the role of documentation in development, there are many, many things you can do to make it do the best for you.

But if there's just one thing you take away from this talk, I hope it's this insight, because it's the one from which everything else follows, that information and documentation are **activities that you engage in**, **not things that you produce**.

THANK YOU

ANY QUESTIONS?

DANIELE PROCIDA

- daniele.procida@divio.com
- EvilDMP on IRC, GitHub, Twitter etc