

Lessons from a 4 million line Django monolith

David Winterbottom

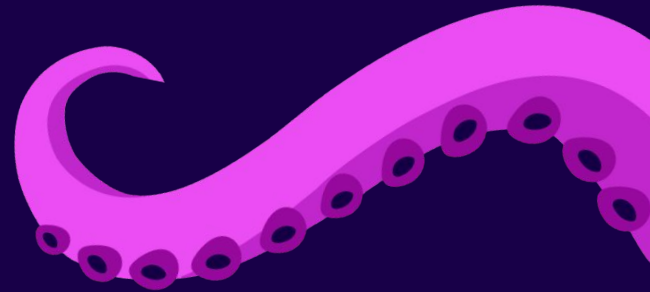


KRAKEN

PART OF THE octopusenergy GROUP



Who are you?



Hello! 🖐️

I'm David

Head of Software Engineering at Kraken Tech

Python / Django

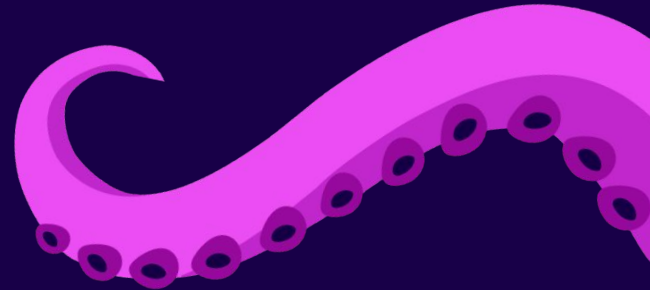
django-oscar

<https://codeinthehole.com>

<https://til.codeinthehole.com>



**4 million line Django
monolith??**



Introducing Kraken



The advanced operating system for utilities

Kraken integrates with all parts of the utilities system. From renewable generation to supplying customers.

Our single operating system helps world-leading businesses across the industry do everything from managing and optimizing energy resources, to delivering excellent customer experiences at a lower cost.




Back in 2015...

Commit

Hello world!

[Browse files](#)

Initial skeleton structure for project in place. This is going to be awesome!

 master

 seboxelsen-disable-db-comms ... 0.1.0



codeinthehole committed on Aug 11, 2015

0 parents commit bb0cb67



300,000 commits later...

● dc132f7 1 minute ago 🕒 315,367 commits



...October 2023

Language	files	blank	comment	code
Python	38060	778586	488703	4099244
JSON	586	22	0	995784
HTML	5321	32246	2811	375913
TypeScript	3183	17018	17402	191477
JavaScript	2221	18148	5233	151329
XML	913	1387	1312	138013
XSD	181	3966	2240	122418
SVG	631	693	77	96378
JSX	975	7295	2429	70518
CSV	345	13	0	51539
Text	2927	13571	0	38392
SCSS	344	7559	2939	36737
GraphQL	349	12024	17910	28513
Markdown	707	10372	78	25710
SQL	438	715	1071	23575



Stats

~530 contributors

~165k tests per CI workflow

>105k pull requests



David Winterbottom  14:59

<https://github.com/octoenergy/kraken-core/pull/107973>

#107973  **Tempjob - terminate meter settings for decommissioned devices**

Now that we are terminating `MeterSettings` when decommissioning devices, adding this tempjob to fix all the devices that were decommissioned previously and will still have active `MeterSettings`



May 18, 2022

Octopus' Kraken Technologies expands into other utilities

By Molly Lempriere



Deepak Ravindran is joining the Octopus Energy Group to run the new business. Image: Kraken Utilities.

Octopus Energy's cloud-native platform Kraken Technologies is expanding to target other utilities, such as water and broadband.

Octopus Energy

+ Add to myFT

Octopus Energy: the UK start-up outgrowing its roots

Founder Greg Jackson puts expansion above profits even as Shell deal turns it into second-biggest supplier

Octopus to buy Shell's household energy firm

🕒 1 September · 💬 Comments



December 23,

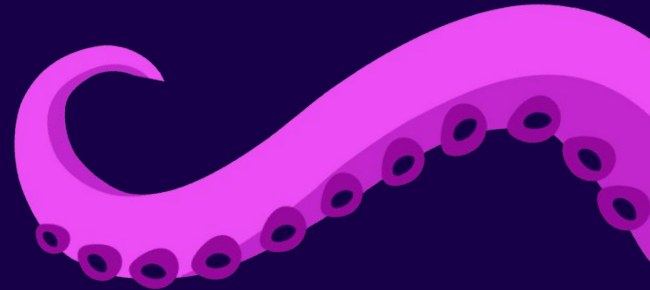
Octopus Energy to take over collapsed supplier Bulb

🕒 29 October 2022

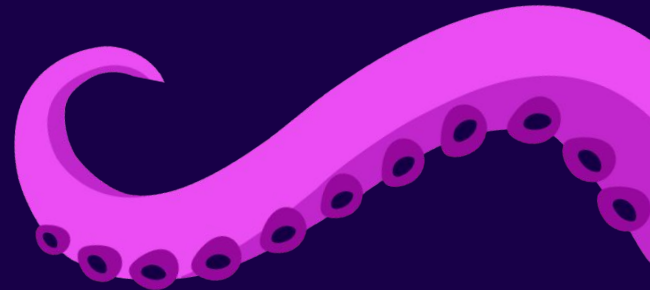
Octopus Energy valuation grows to \$2bn as it expands into Asia

By Molly Lempriere

**What lessons can
you share?**



**How did you build
Kraken to scale?**



We didn't!

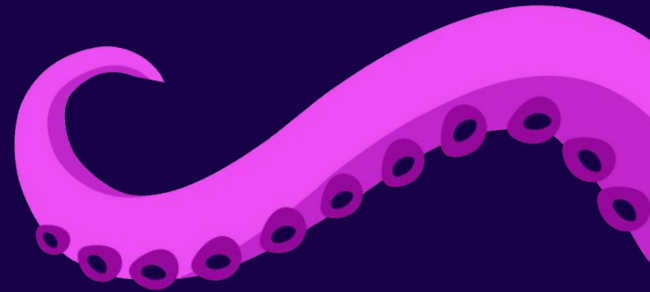
We concentrated on **making it really good** (YAGNI)

We made it **easy to change**

We kept things as **simple** as possible (KISS)



Easy to change?



What is high quality software?

1. Correct, meets requirements
2. Easy to change



Is this easy to CHANGE?

Is this easy
to UNDERSTAND?

Can I change
things SAFELY?

Is the code easy
to READ?

WHY was it done
this way?

Will the test suite
catch regressions?

Can I DEPLOY
changes easily?

Are variables, functions
well named?

Are the commits
self-contained?

Are components loosely
coupled?

Can I deploy changes
in small releases?

Are there comments
and docstrings?

Do commit messages
explain WHY?

Do units follow the
single responsibility principle?

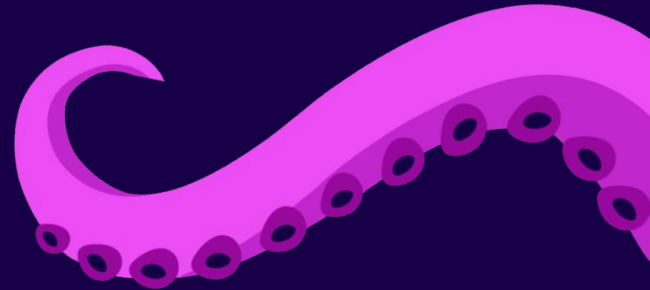
Is the code as SIMPLE
as possible?

Is there a good
audit trail?

Does the code follow
the SOLID principles?



**How to make things
easy to change?**



Conventions



Conventions

Consistency

Good practice

Avoid foot-guns

github.com/octoenergy/conventions

🏠 » / Django

Django

- `CharField` choices
- Class naming conventions
- Model field naming conventions
- Model method naming conventions
- Encapsulate model mutation
- Group methods and properties on models
- Only one table per model
- Create filter methods on querysets, not managers
- Use `.get` when expecting exactly one result
- Don't rely on implicit ordering of querysets
- Don't use audit fields for application logic
- Use a default callable instead of setting `auto_now_add=True`
- Ensure `updated_at` is set when calling `QuerySet.update`
- Be conservative with model `@property` methods
- Ensure `__str__` is unique
- Flash messages
- Avoid model-forms
- Avoid field validators
- Avoid multiple domain calls from an interface component
- Check permissions in dispatch method
- Load resources in `setup` method
- DRF serializers
- Handling out-of-band form errors
- Use database constraints for enforcing data integrity
- Use path converters in URLs instead of regular expressions



Example convention 1

Induced 500s

Ensure your view function can't be induced to return a HTTP 500 response.



```
def my_view(request: http.HttpRequest, **kwargs: str):  
    foo = models.Foo.objects.get(id=kwargs["id"])
```



```
def my_view(request: http.HttpRequest):  
    payload = json.dumps(request.POST)
```



Example convention 2

`__str__` must be unique

Ensure the string returned by a model's `__str__` method uniquely identifies that instance.

Otherwise debugging in Sentry is hard.

```
def __str__(self):  
    return f"#{self.id} ..."
```



How to enforce?

Static analysis

Write custom linters and checks

- Flake8
- Fixit
- Python tests
- Ripgrep?

Run them in CI



Computers are more reliable than humans



Ratchet mechanisms

New convention?



Stop the old way

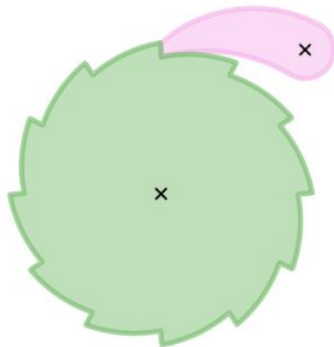


Take your time refactoring for the new way

- Ignore existing violations
- Flake8 ~~has~~ had a `--diff` option



Burndown project!



Application architecture



Example convention 3

Avoid model-based forms, views, ...

Model-based forms/views/...

...are great for shipping CRUD apps quickly...

...but become maintenance problems for large projects with lots of contributors.

Example: Customer sign-up journey

Django doesn't provide a natural home for application logic.



Use cases!

Encapsulate a single operation

Orchestrate:

- Database transactions
- Interacting with other services
- Logging and metrics

!! Agnostic of "interface layer" (i.e. HTTP, CLI concerns)

 Kraken has several interfaces, managed with django-configurations.



Layers!

HTTP
requests



CLI args



RabbitMQ
payload



API site

Management
commands

Celery tasks

INTERFACE LAYER
views, forms, URLs, ...

Dependency
direction

Register new
customer

...

USE CASE LAYER

Application
logic

Save customer
details to DB

Send welcome
email

Create
payment
event

DOMAIN LAYER
Reusable operations

Doesn't know
much about
Django

Account and
customer
models

DATA
LAYER

Send email
with Sendgrid

VENDOR LAYER



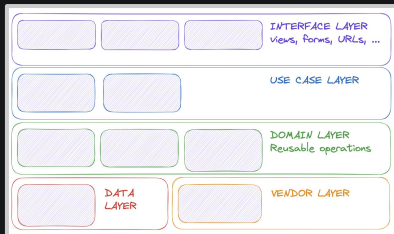
Import linter

```
pip install import-linter
```

Enforces layers

Define contracts for what can import what

```
[importlinter:contract:top-level-layers]
name = Top level layers
type = layers
layers =
    interfaces
    usecases
    domain
    data | vendors
```



README.rst

Import Linter [🔗](#)

pypi [v1.12.0](#) | python [3.8](#) | [3.9](#) | [3.10](#) | [3.11](#) | [3.12](#) | [🔗](#) CI [passing](#)

Import Linter allows you to define and enforce rules for the imports within and between Python packages.

- Free software: BSD license
- Documentation: <https://import-linter.readthedocs.io>.

Overview [🔗](#)

Import Linter is a command line tool to check that you are following a self-imposed architecture within your Python project. It does this by analysing the imports between all the modules in one or more Python packages, and compares this against a set of rules that you provide in a configuration file.

The configuration file contains one or more 'contracts'. Each contract has a specific type, which determines the sort of rules it will apply. For example, the `forbidden` contract type allows you to check that certain modules or packages are not imported by parts of your project.

Import Linter is particularly useful if you are working on a complex codebase within a team, when you want to enforce a particular architectural style. In this case you can add Import Linter to your deployment pipeline, so that any code that does not follow the architecture will fail tests.

If there isn't a built in contract type that fits your desired architecture, you can define a custom one.



Squeezing Django

Then: Django is the framework 🚧

Now: Django is just a library ⚙️



Useful practices



Continuous deployment

Currently: ~150 deploys per day

Small changes - frequent course correction

No unit

200

180

160

140

120

100

80

60

40

20

0

09/23

09/25

09/27

09/29

10/01

10/03

10/05

10/07

10/09

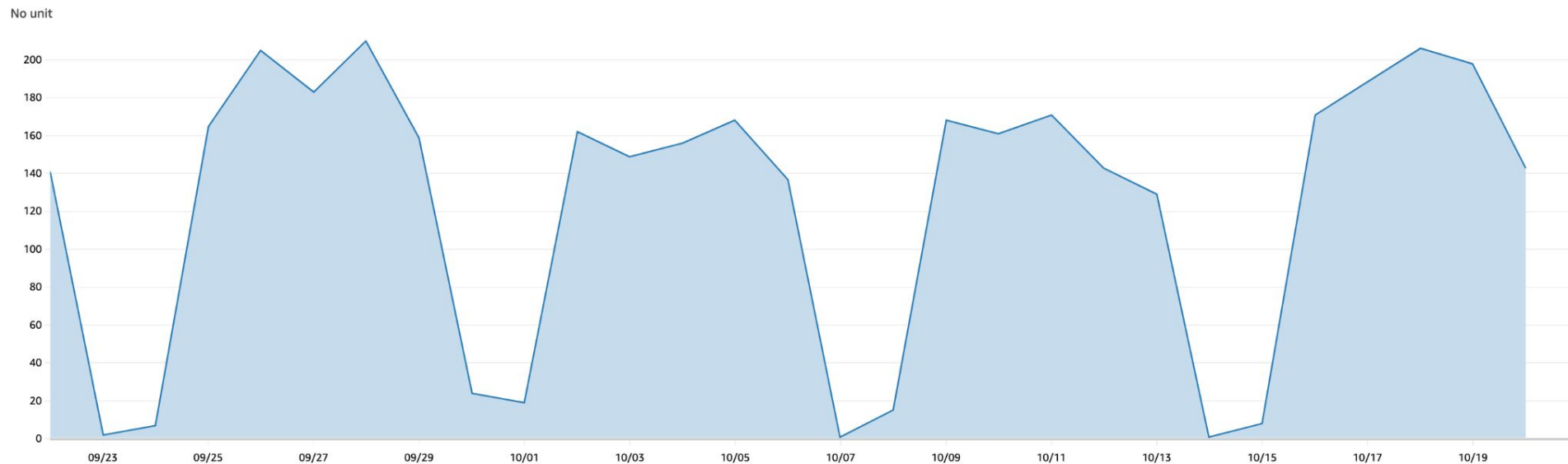
10/11

10/13

10/15

10/17

10/19



End-to-end audit trails

Self-contained commits

- Tests pass
- Any commit could be deployed to production

Rebase before merge

Traverse from line of code to Slack conversation



Document in the deeper place



codeinthehole 1 minute ago



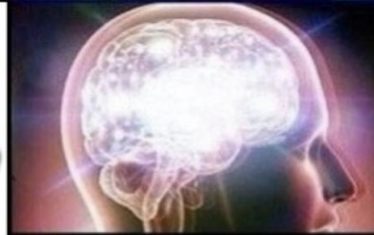
Why did you do this?



**REPLY
IN GITHUB
COMMENTS**



**ADD CODE
COMMENT
EXPLAINING WHY**



**REWORK
CODE TO MAKE
ANSWER OBVIOUS**



imgflip.com








Paying off technical debt





Kraken spa days

Once a month day for:


- Upgrading
- Refactoring
- Speeding up
- Sharpening tools
- Deleting! 🧛


 Kraken spa day 


Friday, 3 November
Monthly on the first Friday


 [Join with Google Meet](#) 


meet.google.com/yyj-xuds-wqq

 [Join by phone](#)
(GB) +44 20 3956 7590 PIN: 388 827 586#

 More phone numbers

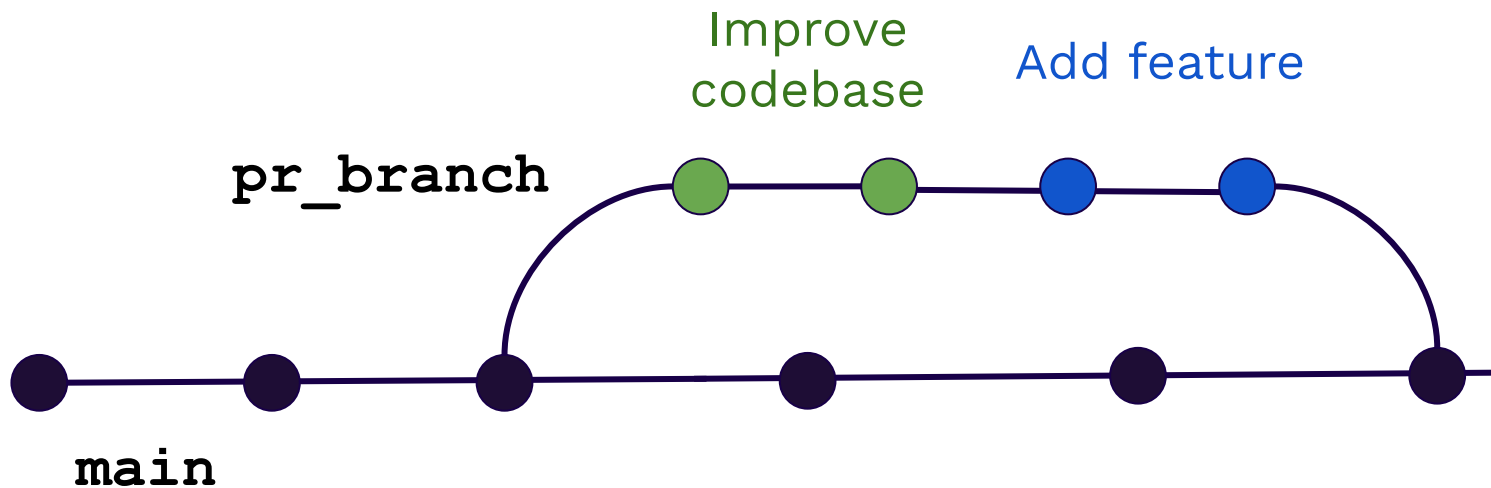
 The full guest list has been hidden because the number of guests is too large.

 David Winterbottom

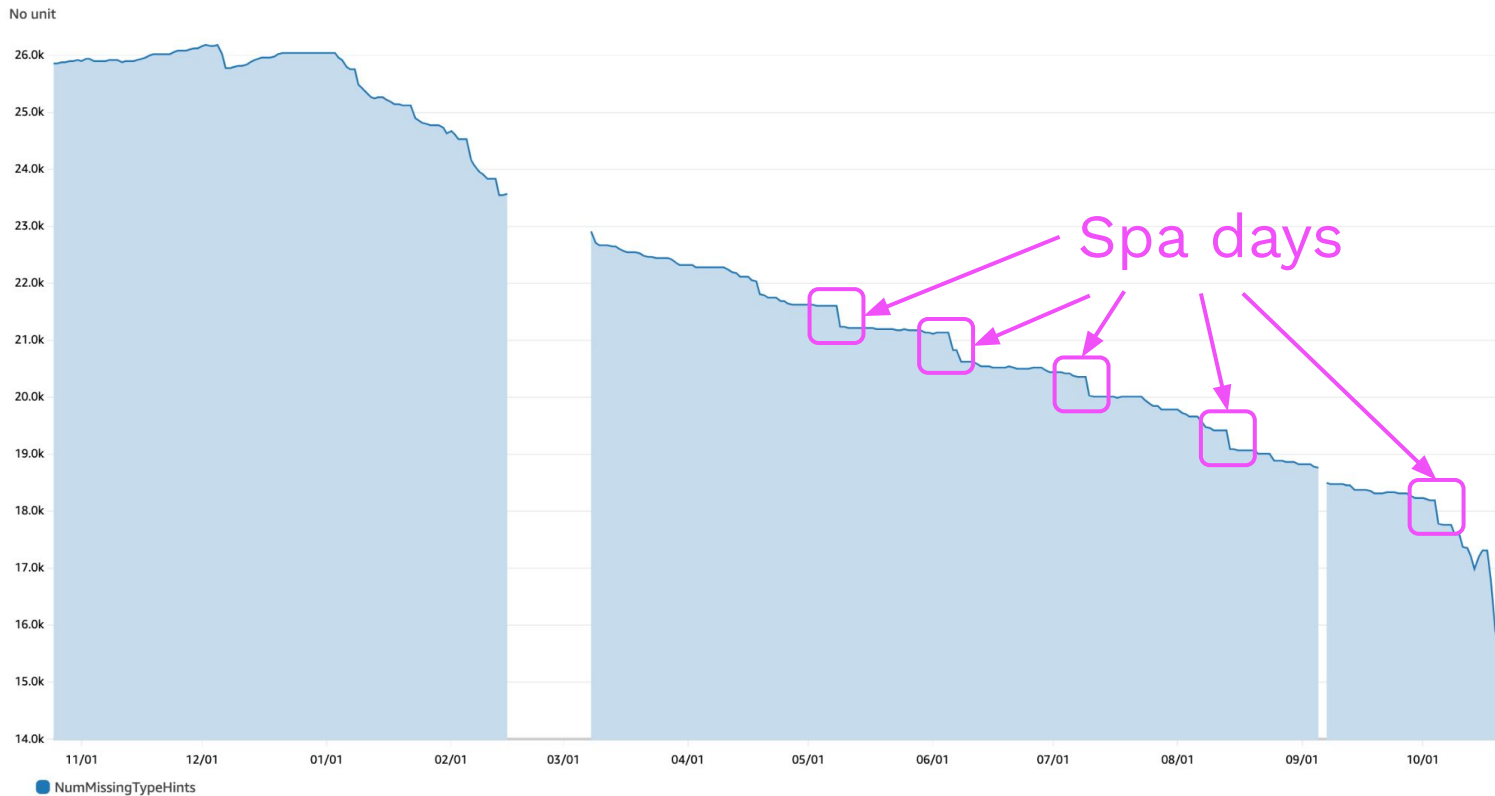
 See the handbook for an overview of spa days.
<https://github.com/octoenergy/handbook/blob/master/events/spa-days.md>



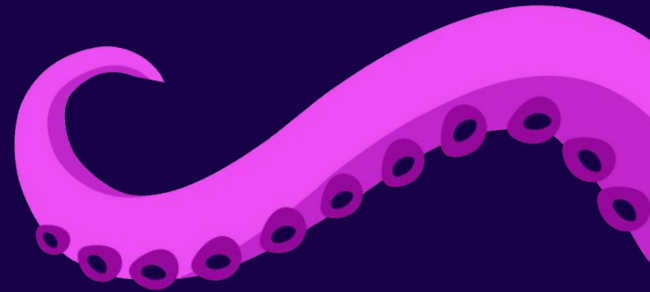
Scouting rule 🏠



Num missing type annotations



Summary



Principle

High quality software is **easy to change**



Monoliths

Large Django projects can be effective

Ensure they are easy to change via:

- Shared conventions
- Use cases
- Layered architecture
- Static analysis and ratchet mechanisms



Useful practices

Continuous deployment

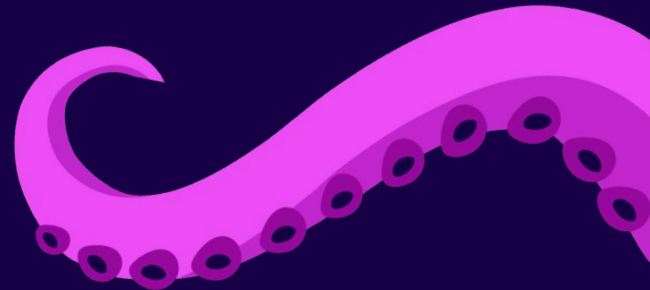
End-to-end audit trails

Documenting in the deeper place

Paying off technical debt in every pull request



Questions?



Bonus content

Problems of a monolith

Flakey tests!

Expensive CI bill

Slow merge-to-deploy speed

Hard to enforce boundaries



Bonus content

What problem are we trying to solve?

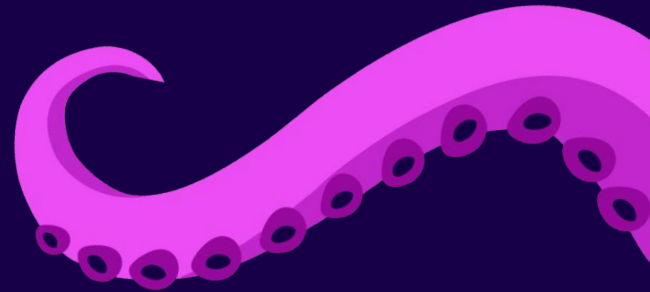
Define success up front

Add instrumentation so we can measure

Verify in production



**Time machine
advice?**



Running out of primary keys

Automatic primary key fields

By default, Django gives each model an auto-incrementing primary key with the type specified per app in `AppConfig.default_auto_field` or globally in the `DEFAULT_AUTO_FIELD` setting. For example:

```
id = models.BigAutoField(primary_key=True)
```

If you'd like to specify a custom primary key, specify `primary_key=True` on one of your fields. If Django sees you've explicitly set `Field.primary_key`, it won't add the automatic `id` column.

Each model requires exactly one field to have `primary_key=True` (either explicitly declared or automatically added).



Set-up linters early

Start projects with strict linting rules

- E.g. Mypy strict mode

