Avoiding Monoliths

DjangoCon EU 2015

Hanna Kollo

About me

- Hanna Kollo
  - Software engineer at Spilgames
- Github
  - https://github.com/spil-hanna
- Twitter
  - https://twitter.com/hannakollo
- Blog
  - http://harbour77.eu/

spilgames

About SpilGames

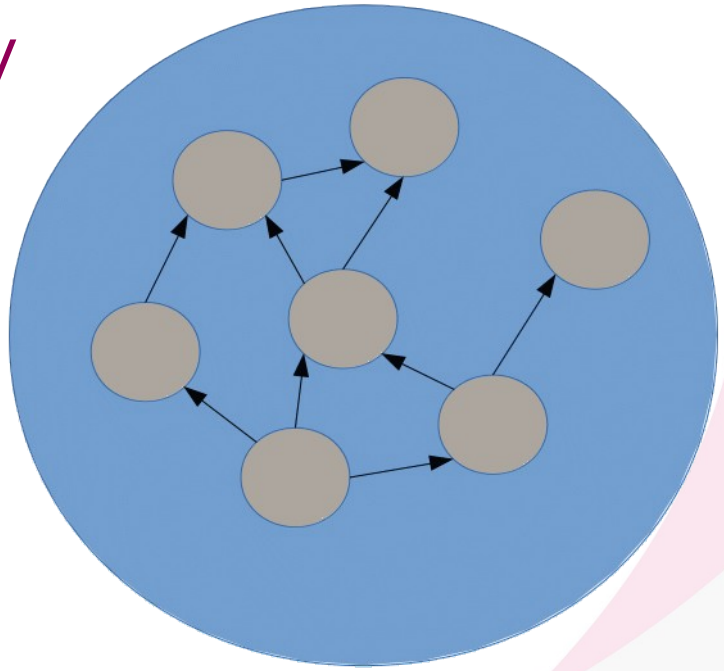
- Casual gaming
- 125 million monthly active users
- ~40 websites
- www.gamesgames.com

Python and Django @ SpilGames

- Service oriented architecture
  - Load-sensitive components in Erlang
  - Some components in Python
- Internal tools
  - Most of them in Python / Django
  - Business-driven applications
    - Changing requirements
    - A challenge to keep the code clean

Monoliths & Django

- Django
  - project
  - apps
- monolith=no modularity
  - one app
  - models, views, html templates
  - Spaghetti code: everything connected to everything



Monoliths (cont)

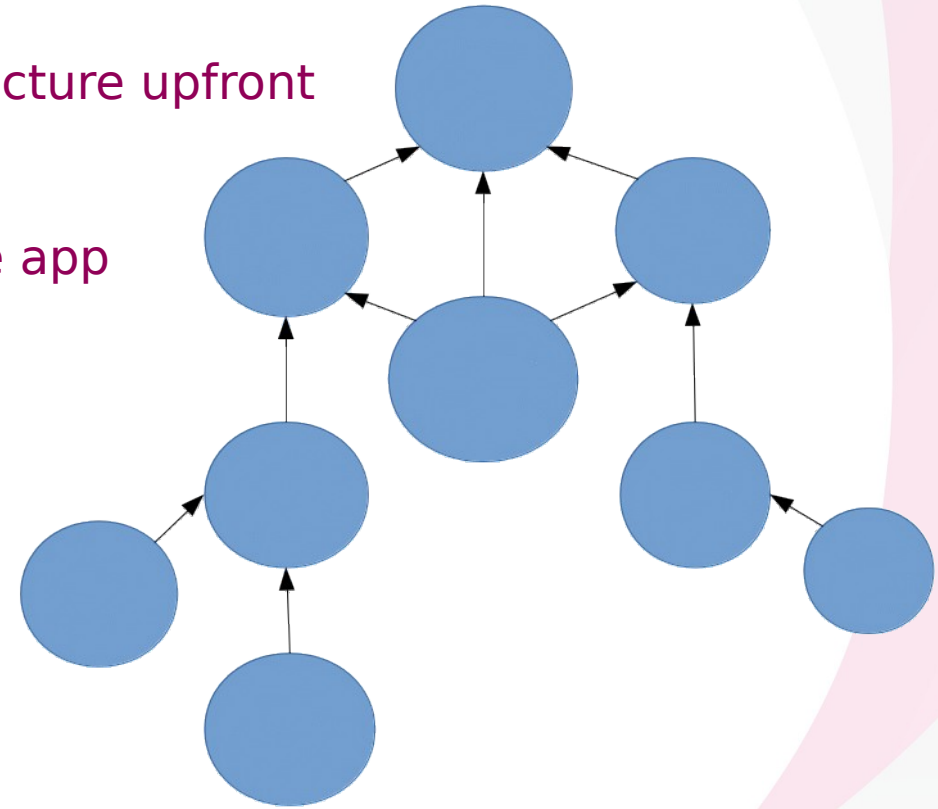
- Monoliths are bad!
  - Opposite of modularity
- Hard
  - Hard to understand
  - Hard to make changes
  - Hard to decide what impacts what
  - Hard to reuse
- Avoiding them
  - The story of the publishing project

Publishing project (1st iter)

- First iteration
  - Design a modular structure upfront
  - Many small apps
    - Few models in one app

Publishing project (1st iter)

- First iteration
  - Design a modular structure upfront
  - Many small apps
    - Few models in one app
- No monolith
  - But...

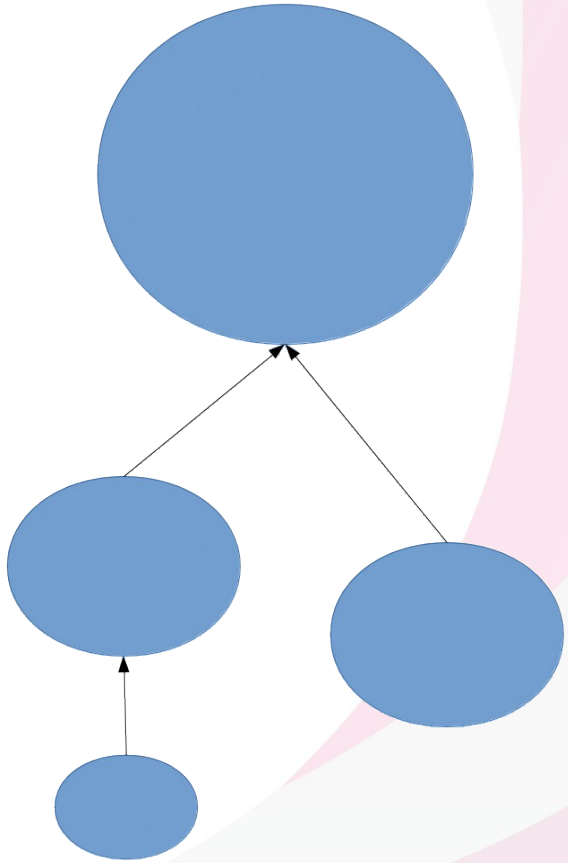


Publishing project (2nd iter)

- Start with one app
  - Get it to a stable point (minimum viable product)
- Add more apps
  - One by one
  - Consider major refactoring

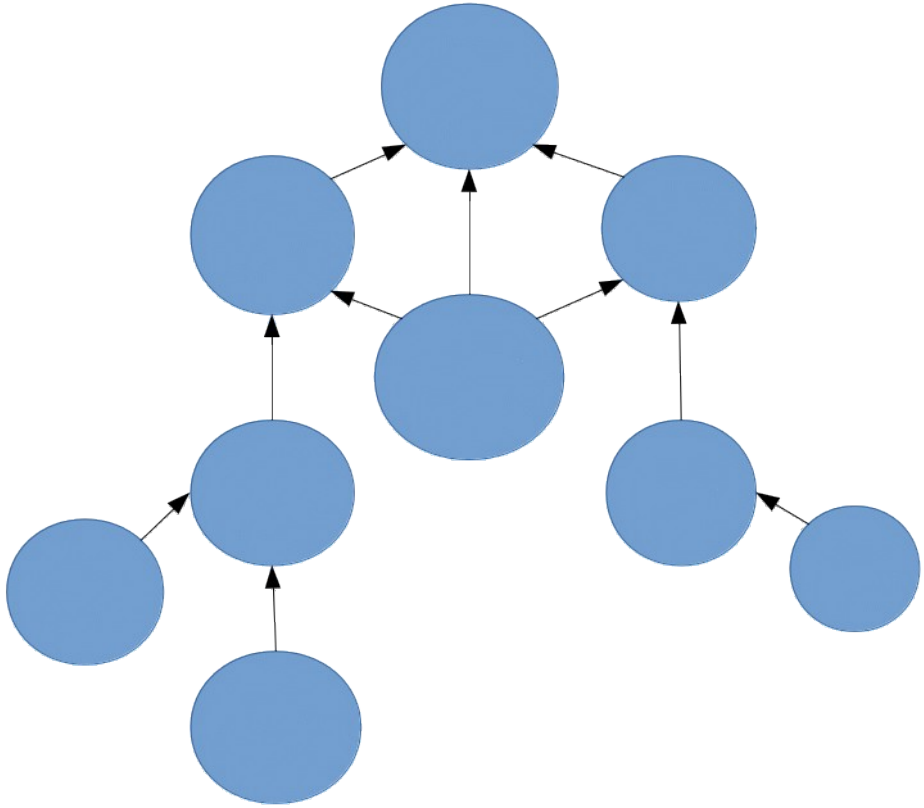
Publishing project (2nd iter)

- Start with one app
  - Get it to a stable point (minimum viable product)
- Add more apps
  - One by one
  - Consider major refactoring
- Few large apps
  - Many models in one app
  - Tree-like structure of dependencies



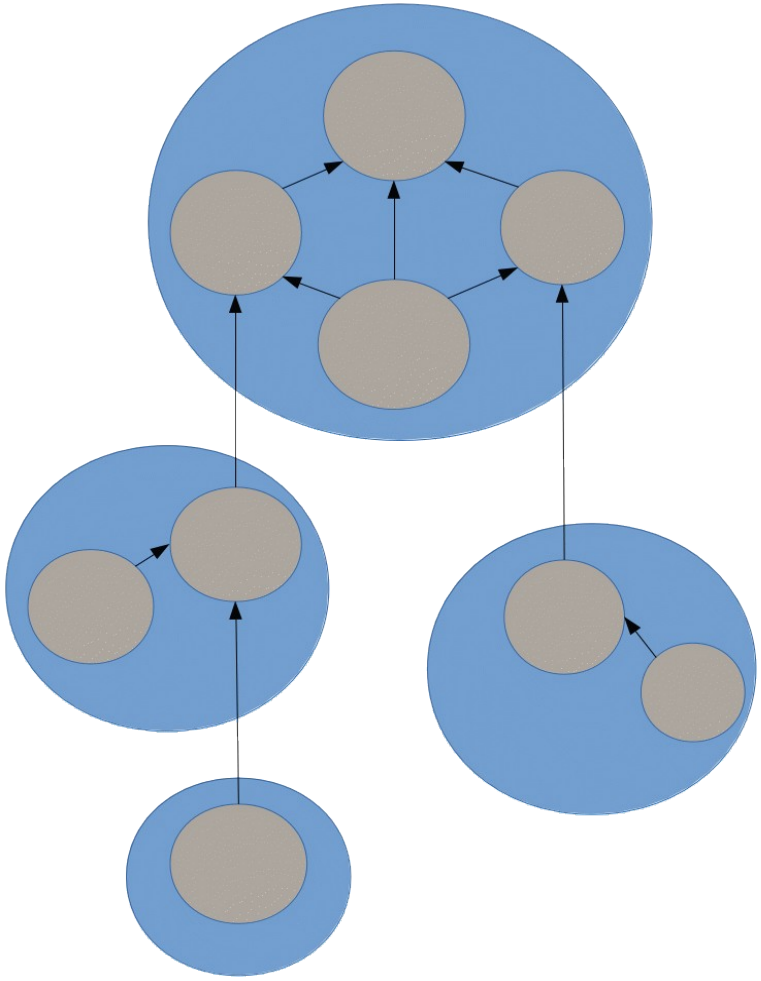
spilgames

Transition



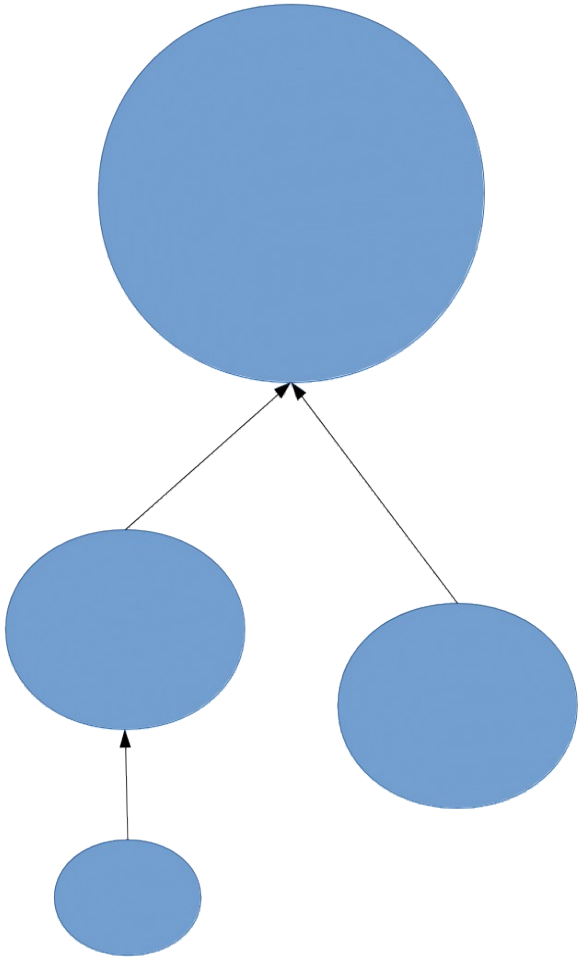
spilgames

Transition



spilgames

Transition



spilgames

Why is it better

- No monolith
  - No spaghetti code
  - Facilitates growth
- Main structural difference
  - 1st iter: Few models in many apps
  - 2nd iter: Many models in few app
- Driving force when adding code
  - Business logic
- Problems with the size?

Structure within the app

- Where to put business logic?
  - Common practice: Model managers
  - Our practice: Services layer
- Services
  - Business logic... is not only about data
  - Examples
    - export some data somewhere
    - send notification emails
    - “HAL, open the pod bay doors”
  - Easier to unit-test

Using services

```
from models import PodBayDoor
from services import HAL

def view1(request):
    # read object directly
    door = PodBayDoor.objects.get(pk=1)
    # read object through service
    info = HAL.check_state(door)

    # update object through service
    if request.method == 'POST':
        HAL.open(door)
```

Summary

- Two options
  - Few apps with many models ← better
  - Many apps with few models
- Start with one app
  - grow into a tree-like structure of apps
- Services layer
  - For business logic