2017

odoo

EXPERIENCE

Inherit Odoo Mixins to Empower your App

Let us recode everything from scratch. Or not.

Thibault DELAVALLEE  • R&D Engineer

based on work from Damien BOUVY

If only I knew there were so cool mixins in Odoo, I would have avoided a lot of issues. To future generations: reuse, do not recode !

— Abraham Lincoln

Some common features

- Communication & Organization
  - discuss on a document, send mailings
  - schedule activities
- Marketing
  - Get customer satisfaction insights
  - Track campaigns and mediums
- Website
  - Manage document publication
  - Promote pages

Some common features

- Communication & Organization
  - discuss on a document, send mailings
  - schedule activities

- Marketing
  - Get customer satisfaction insights
  - Track campaigns and mediums

- Website
  - Manage document publication
  - Promote pages

Some common features

- Communication & Organization
  - discuss on a document, send mailings
  - schedule activities

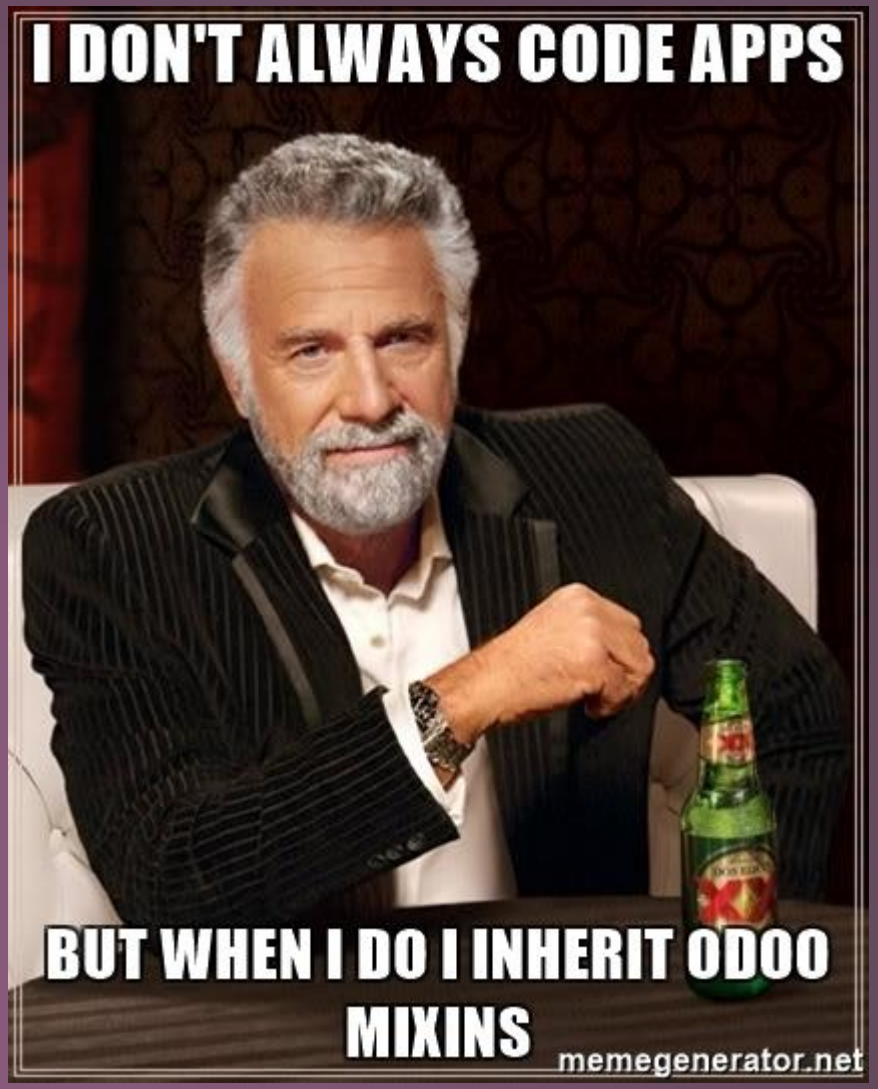
- mail.thread
- mail.activity.mixin

- Marketing
  - Get customer satisfaction insights
  - Track campaigns and mediums

- rating.mixin
- utm.mixin

- Website
  - Manage document publication
  - Promote pages

- portal, website.published
- website.seo.metadata



— A Classy Cool Dev

Thanks Classy Cool Dev!
But what is a Mixin ?

Mixin Class: vaziztasse ?

- Mixin class: extracts transversal features
  - AbstractModel: no table, only for definition
  - offer features through inheritance
  - e.g. messaging mechanism, customer satisfaction request, ...

```
class MailThread(models.AbstractModel):
    _name = 'mail.thread'
    _description = 'Mail Thread Mixin'

    message_ids = fields.One2many(
        'mail.message', 'Messages')
    message_follower_ids = fields.One2many(
        'mail.followers', 'Followers')

    def message_post(self, body):
        # do stuff

    def message_subscribe(self, partners):
        # do stuff
```

Mixin Class: vaziztasse ?

- Mixin class: extracts transversal features
- Inheritance
  - “copy” fields on child

```
class MailThread(models.AbstractModel):
    _name = 'mail.thread'

    message_ids = fields.One2many(...)
    message_follower_ids = fields.One2many(...)
```

```
class Plant(models.Model):
    _inherit = ['mail.thread']
    name = fields.Char('Plant Name')
```

```
class Order(models.Model):
    _inherit = ['mail.thread']
    name = fields.Char('Reference')
```

Mixin Class: vaziztasse ?

- Mixin class: extracts transversal features
- Inheritance
  - “copy” fields on child

```
class MailThread(models.AbstractModel):
    _name = 'mail.thread'

    message_ids = fields.One2many(...)
    message_follower_ids = fields.One2many(...)
```

```
class Plant(models.Model):
    _inherit = ['mail.thread']

    name = fields.Char('Plant Name')
    message_ids = fields.Many2one(...)
    message_follower_ids = fields.One2many(...)
```

```
class Order(models.Model):
    _inherit = ['mail.thread']

    name = fields.Char('Reference')
    message_ids = fields.Many2one(...)
    message_follower_ids = fields.One2many(...)
```

Mixin Class: vaziztasse ?

- Mixin classes: extract transversal features
- Inheritance
  - “copy” fields on child
  - class inheritance: methods, super(), ...

```python
class MailThread(models.AbstractModel):
    _name = 'mail.thread'

    message_ids = fields.One2many(...)
    message_follower_ids = fields.One2many(...)

    def message_post(self, body):
        # do stuff
        return message
```

```python
class Plant(models.Model):
    _name = 'plant.plant'
    _inherit = ['mail.thread']

    name = fields.Char('Plant Name')
    price = fields.Integer('Plant Price')

    def say_hello(self):
        self.message_post('Hello')

    def message_post(self, body):
        if self.message_ids: ...
        self.message_follower_ids.write({})
        return super()
```

Mixin Class: vaziztasse ?

- Mixin classes: extract transversal features
- Inheritance
  - “copy” fields on child, class inheritance: methods, super(), ...
  - **improved in v11: inherit abstract class itself**

    -> tweaking abstract through inheritance applies to all childs

```
class MyOwnMailThread(models.AbstractModel):
    _name = 'mail.thread'
    _inherit = 'mail.thread'

    message_my_own = fields.Integer(...)

    def message_my_own_stuff(self):
        ...
        return

    def message_post(self, body):
        # do more stuff
        self.message_my_own_stuff()
        return super()
```

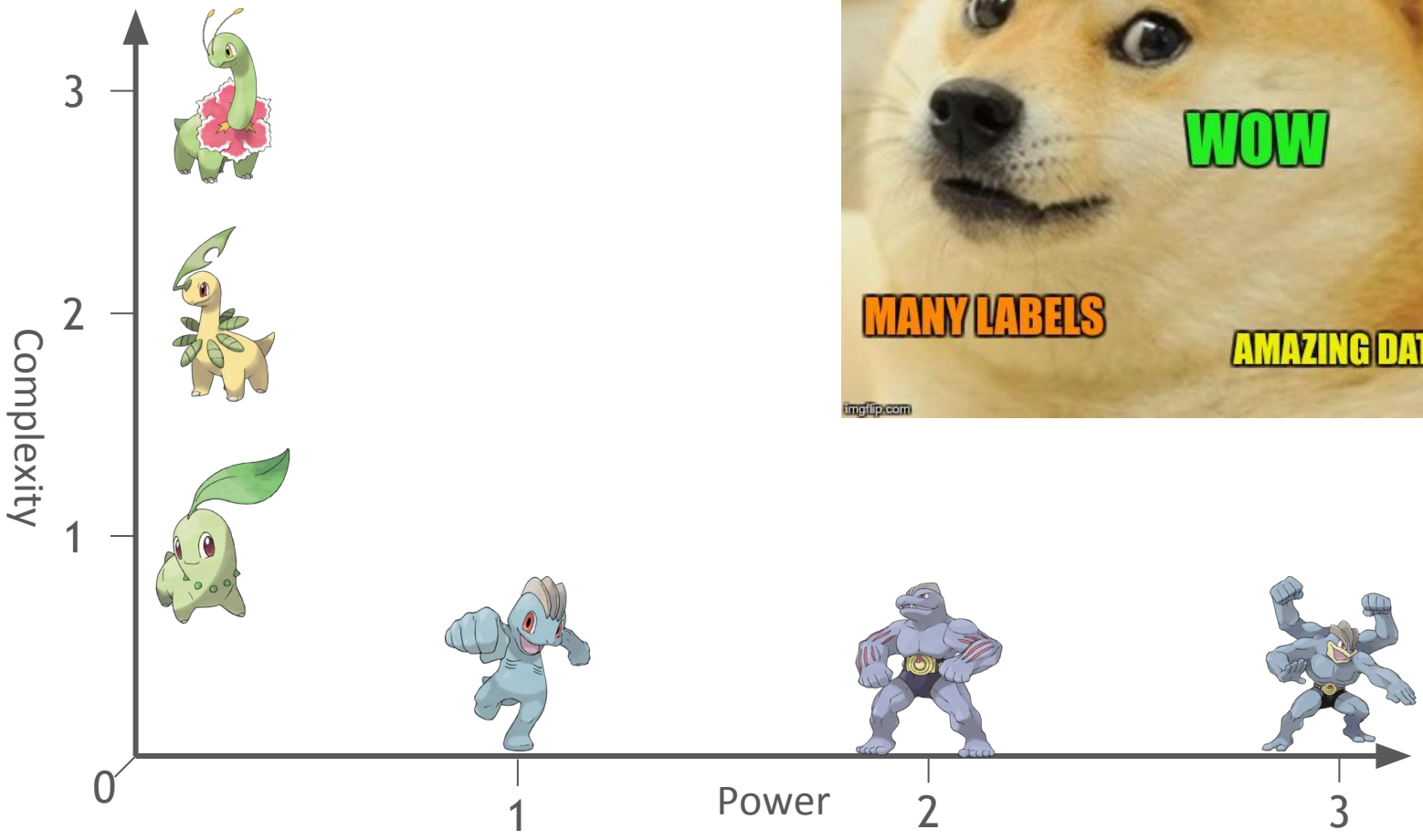
I master mixins. Now what ?

The graph


SO SCIENTIFIC
MUCH GRAPH
WOW
MANY LABELS
AMAZING DATA
imgflip.com

Complexity
3
2
1
0
1
Power
2
3

The graph


Complexity
3
2
1
0
Power
1
2
3

SO SCIENTIFIC
MUCH GRAPH
WOW
MANY LABELS
AMAZING DATA
imgflip.com

Methodology

- Features of mixins
- How to implement them
- Code bits
- Live result
- Documentation ?

Methodology

- Features of mixins
- How to implement them
- Code bits
- Live result
- Documentation ?
  - well …
  - Odoo doc + code
  - Source code of demo App

2 Communication and Organization

Mail Thread basics

- Add messaging to any class
- Auto-interfacing with e-mail
- How to
  - inherit mail.thread
  - add widgets
  - have fun !

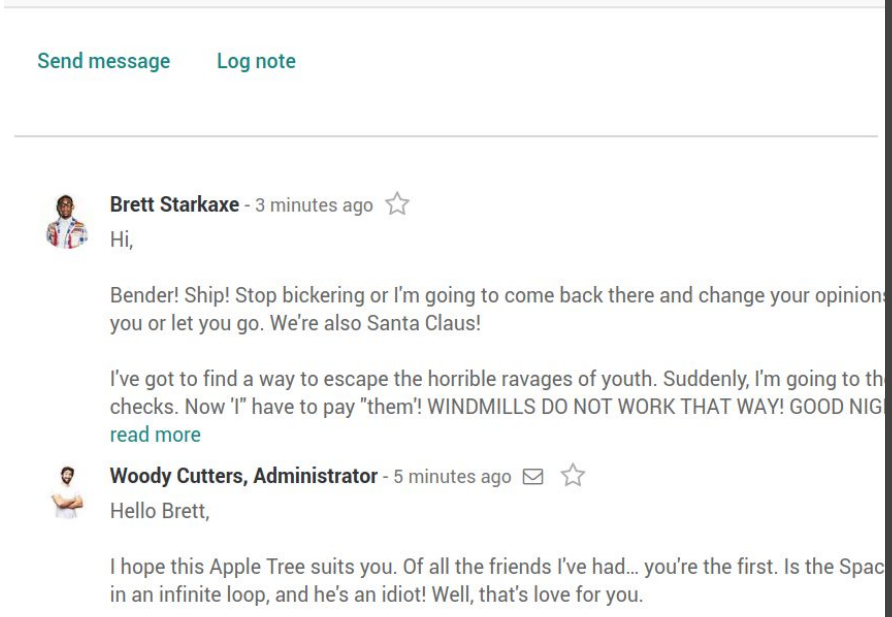
```
class Plant(models.Model):
    _name = 'plant.plant'
    _description = 'Plant'
    _inherit = ['mail.thread']
```

```
<div class="oe_chatter">
    <field name="message_follower_ids"
        widget="mail_followers"/>
    <field name="message_ids"
        widget="mail_thread"/>
</div>
```

```
{
       'name': Plant Nursery,
       'depends': ['mail'],
}
```

Mail Thread: chatter

- Messages linked to a document

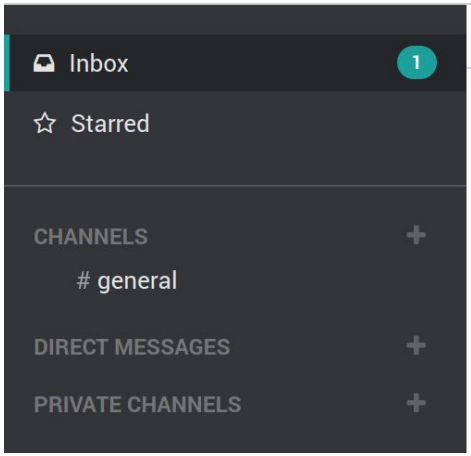



```python
class Message(models.Model):
    _name = 'mail.message'
    _description = 'Message'

    model = fields.Char(...)
    res_id = fields.Integer(...)

    notified_partner_ids = fields.Many2Many(...)

    def create(self, values):
        msg = super()
        msg._notify()
        return msg

    def _notify(self):
        followers = self.get_followers()
        followers.notify()
        followers.send_mail()

class Plant(models.Model):
    _inherit = ['mail.thread']

    # coming from mail.thread inheritance
    message_ids = fields.One2many(...)
```

Mail Thread: chatter

- Messages linked to a document
- Communication through Chatter

```python
class MailThread(models.AbstractModel):
    _name = 'mail.thread'

    message_ids = fields.One2many(...)

    def message_post(self, subject, body, **kw):
        self.env['mail.message'].create({
            'model': self.model,
            'res_id': self.res_id})

class Plant(models.Model):
    _name = 'plant.plant'
    _inherit = ['mail.thread']

    def message_post(self, subject, body, **kw):
        super()
```



Today

Brett Starkaxe - 17 minutes ago on Apple Tree

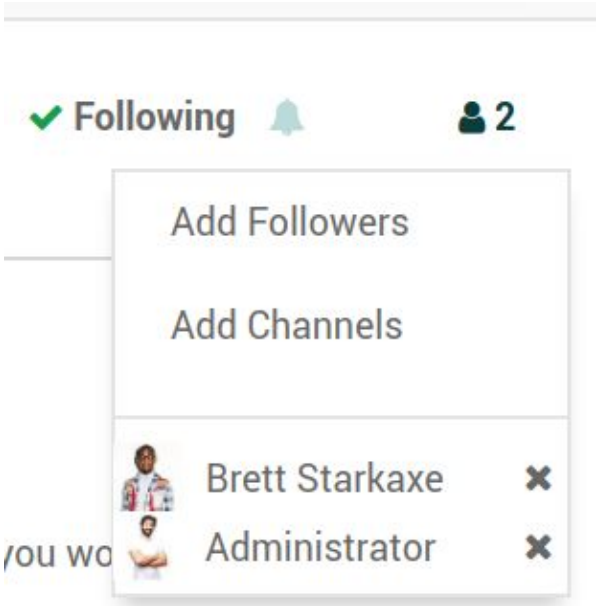
Hi,

Bender! Ship! Stop bickering or I'm going to come back there and change your opinions manually! You know the worst thing about being a slave? They make you work, but they don't pay you or let you go. We're also Santa Claus!

I've got to find a way to escape the horrible ravages of youth. Suddenly, I'm going to the bathroom like clockwork, every three hours. And those jerks at Social Security stopped sending me checks. Now 'I" have to pay "them'! WINDMILLS DO NOT WORK THAT WAY! GOOD NIGHT!

Mail Thread: chatter

- Messages linked to a document
- Communication through Chatter
- Followers management




```python
class MailThread(models.AbstractModel):
    _name = 'mail.thread'

    message_follower_ids = fields.One2many(...)

    def message_subscribe(self, partners, channels):
        # add followers and listeners
        Follower.create(partners)

class Plant(models.Model):
    _name = 'plant.plant'
    _inherit = ['mail.thread']

    def message_subscribe(self, partners, channels):
        super()
```

Mail Thread: mailgateway

- Process and route incoming emails
- Ensure thread detection
- App-specific behavior on new thread or on update

```python
class MailThread(models.AbstractModel):
    _name = 'mail.thread'

    def message_process(self, email):
        # process email values

    def message_route(self, message):
        # heuristics when incoming email detected


class Plant(models.Model):
    _name = 'plant.plant'
    _inherit = ['mail.thread']

    def message_new(self, message):
        # do sthg when creating from email
        super()

    def message_update(self, message):
        # do sthg when incoming email
        super()
```

Mail Activity

- Activities management on document
- Interfacing with Chatter
- How to
  - inherit mail.activity.mixin
  - add widgets
  - have much fun !

```python
class Plant(models.Model):
    _name = 'plant.plant'
    _description = 'Plant'
    _inherit = ['mail.thread', 'mail.activity.mixin']
```

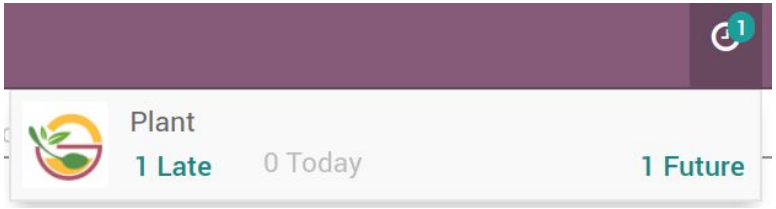
```xml
<div class="oe_chatter">
    <field name="message_ids" widget="mail_thread"/>
    <field name="activity_ids" widget="mail_activity"/>
</div>
```

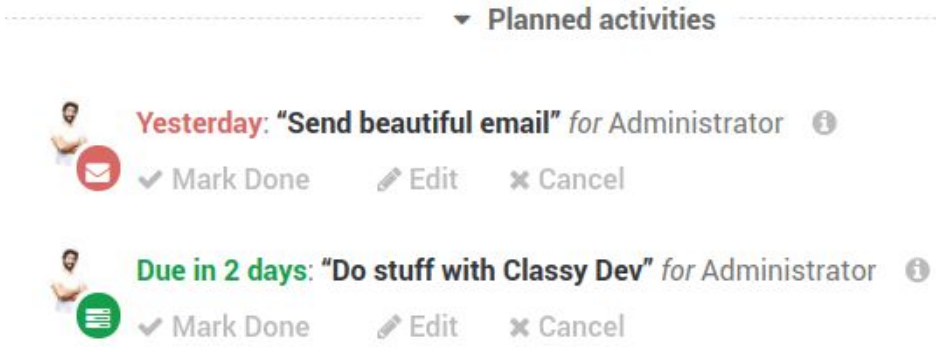
```
{
       'name': Plant Nursery,
       'depends': ['mail'],
}
```

Mail Activity

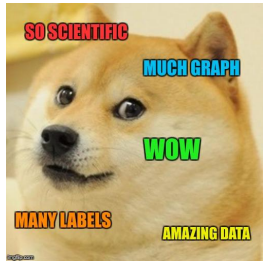
- Activities management on document
- Activity-based state
- Filters

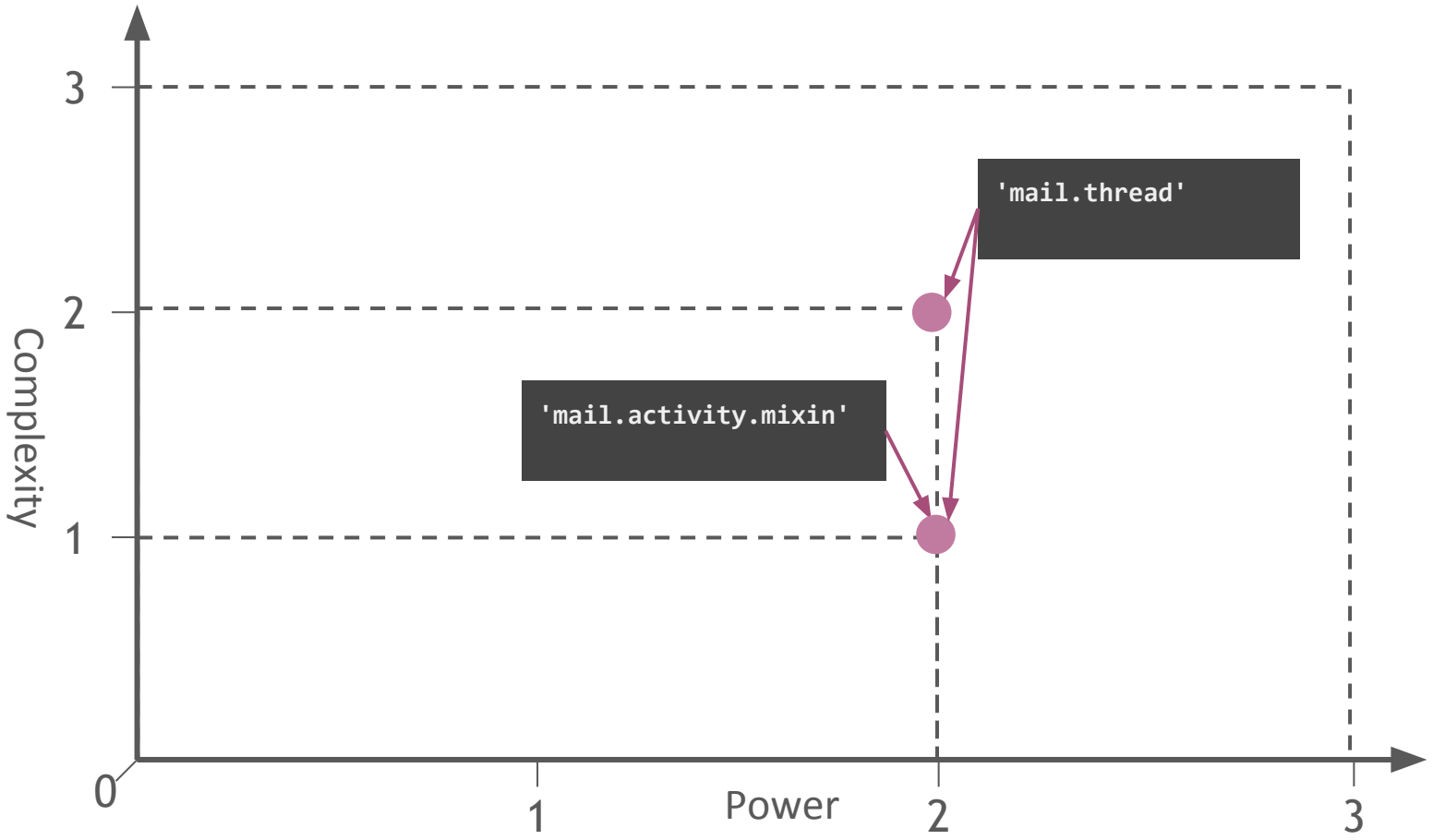
```python
class Plant(models.Model):
    _name = 'plant.plant'
    _description = 'Plant'
    _inherit = ['mail.thread', 'mail.activity.mixin']

    # coming from mail.activity inheritance
    activity_ids = fields.One2many('mail.activity')
    activity_state = fields.Selection([
        ('overdue', 'Overdue'),
        ('today', 'Today'),
        ('planned', 'Planned')])
```





The graph


SO SCIENTIFIC
MUCH GRAPH
WOW
MANY LABELS
AMAZING DATA


3
2
1
0
Complexity
1
Power
2
3
'mail.thread'
'mail.activity.mixin'

3

Customer Satisfaction and UTM

Rating

- Add rating on any class
- Request rating via email/route
- Analyse your ratings

```python
class Plant(models.Model):
    _name = 'plant.plant'
    _description = 'Plant'
    _inherit = ['mail.thread', 'rating.mixin']
```

```
{
       'name': Plant Nursery,
       'depends': ['rating']
}
```

RATING REQUEST

Thanks! We appreciate your fe

Your rating has been submitted.

you are **satisfied**
on our services on "**Gretta GmailAddress**"
by **Power User**.

Rating: a bit of work

- Rated partner (user_id.partner_id field)
  - rating_get_rated_partner_id
- Customer (partner_id field)
  - rating_get_partner_id
- Access token
  - rating_get_access_token
- Routes (/rating/<token>/<score>)
- Email Template using this data

```
class Order(models.Model):
    _name = 'plant.order
    _inherit = ['mail.thread', 'rating.mixin']

    def rating_get_partner_id(self):
        return self.customer_id
```

```
<record model="mail.template"
        id="mail_template_plant_order_rating">
    <field name="name">Plant: Rating Request</field>
    <field name="email_from">
        ${(object.rating_get_rated_partner_id().email or '')
| safe}
    </field>
...
</record>
```

Rating: a bit of work

Today

Woody Cutters, Administrator - now

Satisfaction Survey

Hello,

Please take a moment to rate our services related to the order "**Gimli Order**" assigned to **Administrator**.

We appreciate your feedback. It helps us to improve continuously.

Tell us how you feel about our service:

(click on one of these smileys)

read more

Email automatically sent by Odoo Plant Nursery for Woody Cutters

UTM

- Track incoming visitors
- Pre-built with three fields: campaign, source, medium
- Simple to extend

```
class Plant(models.Model):
    _name = 'plant.plant'
    _description = 'Plant'
    _inherit = ['utm.mixin']
```

```
    campaign_id = fields.Many2one('utm.campaign')
    source_id = fields.Many2one('utm.source')
    medium_id = fields.Many2one('utm.medium')
```

```
{
       'name': Plant Nursery,
       'depends': ['utm'],
}
```

UTM

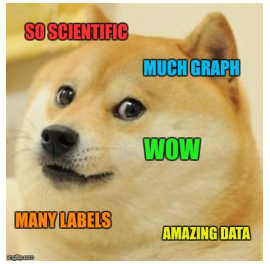
- Track incoming visitors
- Pre-built with three fields: campaign, source, medium
- Simple to extend
- URL parsing

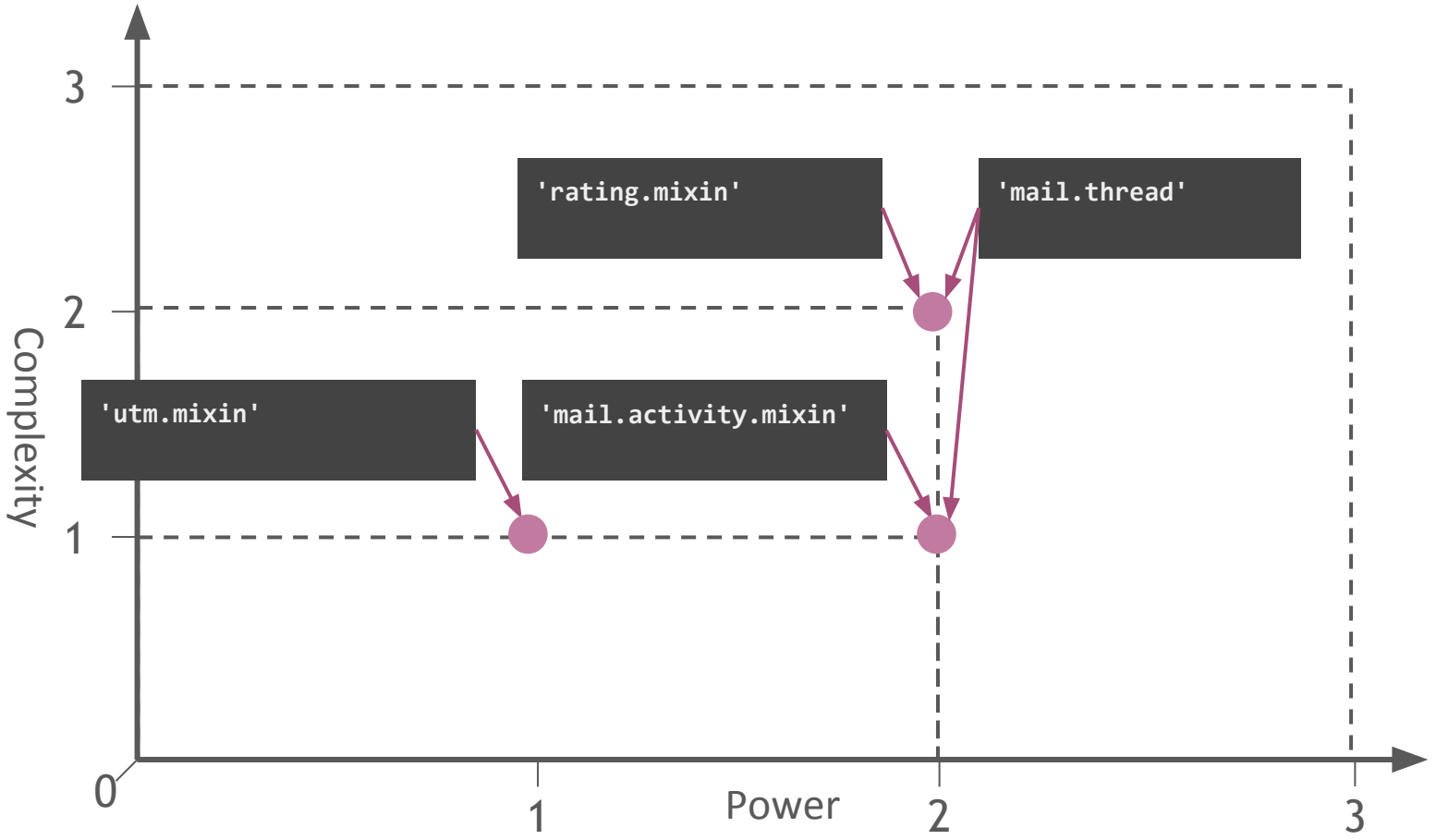
```
class Plant(models.Model):
    _name = 'plant.plant'
    _description = 'Plant'
    _inherit = ['utm.mixin']
```

https://www.odoo.com?campaign_id=sale&medium=facebook&source_id=facebook_ads

Campaign	Sale
Source	Facebook
Medium	Facebook Ads

The graph


SO SCIENTIFIC
MUCH GRAPH
WOW
MANY LABELS
AMAZING DATA


3
2
1
0
Complexity
'rating.mixin'
'mail.thread'
'utm.mixin'
'mail.activity.mixin'
1
Power
2
3

4

Website, Portal and Pages

SEO

- ‘Optimize’ SE rankings
- Add fields
  - Page title
  - Keywords
  - Description

```
class Plant(models.Model):
    _name = 'plant.plant'
    _description = 'Plant'
    _inherit = ['website.seo.metadata']
```

```
website_meta_title = fields.Char('')
website_meta_description = fields.Text('')
website_meta_description = fields.Char('')
```

```
{
       'name': Plant Nursery,
       'depends': ['website'],
}
```

SEO: Website

- ‘main_object’ magic in website
- Website promote button
- Set SEO metadata

```
@route('/plant/<model(“plant.plant”):plant>/',
       type='http',
       auth='public,
       website=True)
def plant(self, plant, **post):
    values = {
        'main_object': plant,
        'plant': plant,
        'stuff': post.get('stuff', 'brol'),
    }
    return request.render("plant_page", values)
```

SEO: Website

Promote This Page Get this page efficiently referenced in Google to attract more visitors.

Page Title Apple Tree | My Website

Description Super tree

Preview how your page will be listed on Google

Apple Tree | My Website
http://localhost:8069/plant/apple-tree-7
Super tree

Define Keywords describing your page content

Keyword | English | Add

apple tree

Suggested

Most searched topics related to your keywords, ordered by importance:

honda | s for sale | dental | diseases | leaves | inn | yard | daycare | learning center

Legend: Not used | In title | In description | In page's content

Save | Discard

Publish

- Control visibility of documents
- Add fields
  - published
  - URL (slug)
- **Access rights & route management still up to you !**

```
class Plant(models.Model):
    _name = 'plant.plant'
    _description = 'Plant'
    _inherit = ['website.published.mixin']
```

```
{
      'name': Plant Nursery,
      'depends': ['website'],
}
```

```
from odoo.ir_http import slug

website_published = fields.Boolean('')
website_url = fields.Char(compute='_compute_website_url')

def _compute_website_url(self):
    for plant in self:
        plant.website_url = '/plants/%s' % slug(self)
```

Publish: backend

- Control visibility of documents
- Wesite URL: computed field

  -> slug (/plant/tree-1)
- Backend: use website_button

  -> jump to website_url


```python
class Plant(models.Model):
    _name = 'plant.plant'
    _description = 'Plant'
    _inherit = ['website.published.mixin']
```

```xml
<button class="oe_stat_button"
        name="website_publish_button"
        type="object" icon="fa-globe">
    <field name="website_published"
           widget="website_button"/>
</button>
```

```python
from odoo.ir_http import slug

website_published = fields.Boolean('')
website_url = fields.Char(compute='_compute_website_url')

def _compute_website_url(self):
    for plant in self:
        plant.website_url = slug(self)
```

Publish: frontend

- publish_management widget
- Link backend / frontend

  -> give action to execute

```
class Plant(models.Model):
    _name = 'plant.plant'
    _description = 'Plant'
    _inherit = ['website.published.mixin']
```

```
<t t-call="website.publish_management">
    <t t-set="object" t-value="plant"/>
    <t t-set="publish_edit" t-value="True"/>
    <t t-set="action"
        t-value="'plant_nursery.plant_plant_action'"/>
</t>
```

Portal

- Document- and App- specific portal url computation
- Generic URL computation
  - /mail/view controller
  - access_token
  - partner_id
  - integration with auth_signup

```
class Order(models.Model):
    _name = 'plant.order'
    _description = 'Order'
    _inherit = ['portal.mixin']
```

```
{
      'name': Plant Nursery,
      'depends': ['portal']
}
```

```
class PortalMixin(models.AnstractModel):
    _name = 'portal.mixin'

    def get_share_url(self, body):
        # compute generic /mail/view URL
        return '/mail/view' ...
```

Portal

- Document- and App- specific portal url computation
- Generic URL computation
- In your App
  - access_token definition
  - specific portal_url computation

```
class Order(models.Model):
    _name = 'plant.order'
    _description = 'Order'
    _inherit = ['portal.mixin']

    access_token = fields.Char('Access Token')

    def _compute_portal_url(self, body):
        self.portal_url = '/order/%s/%s' % (self.id,
self.access_token)
```

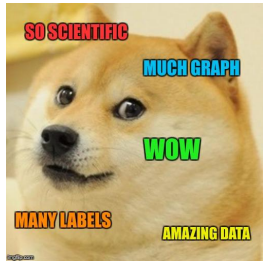
```
class PortalMixin(models.AnstractModel):
    _name = 'portal.mixin'

    def get_share_url(self, body):
        # compute generic /mail/view URL
```

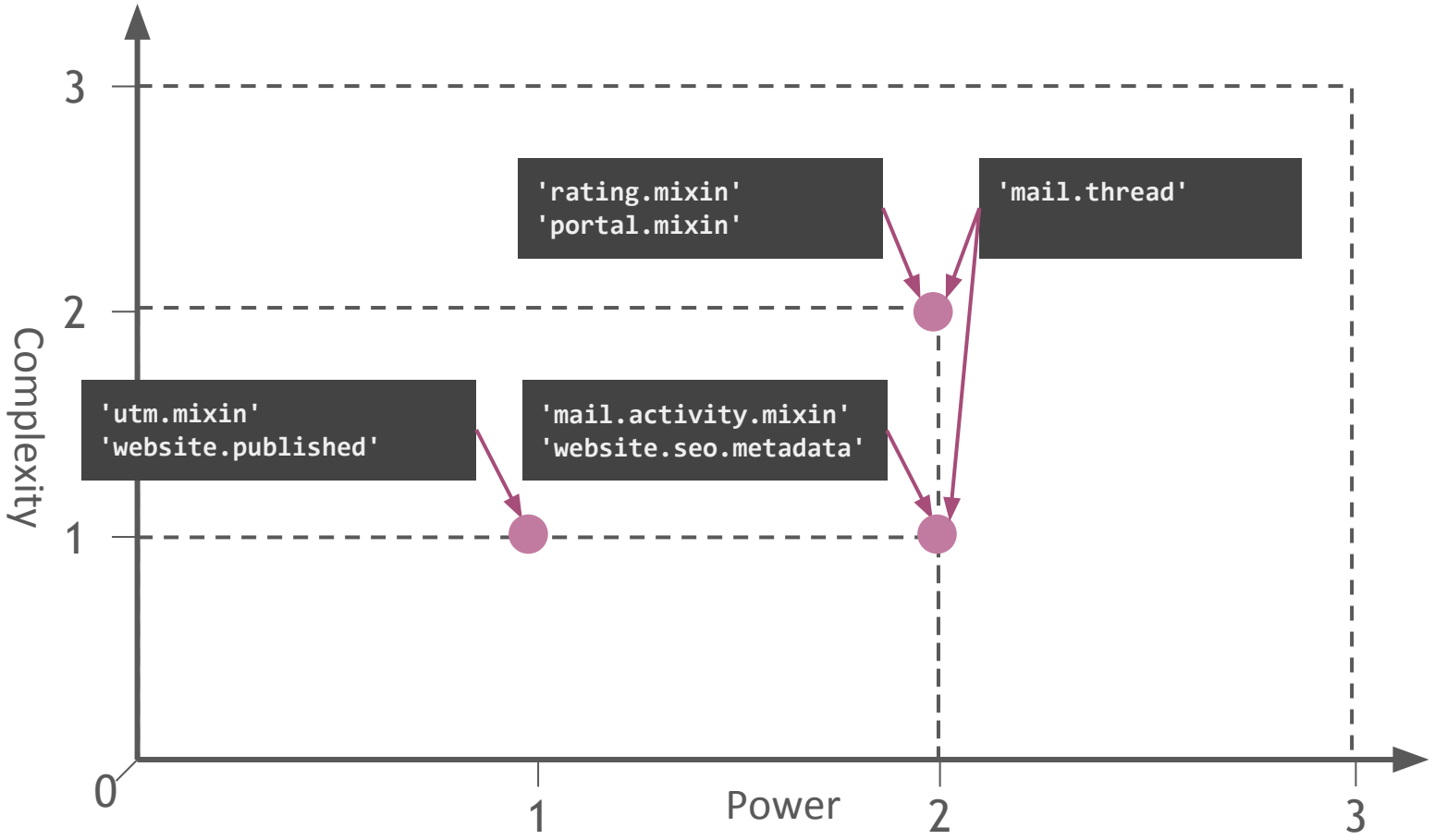
Portal

- Document- and App- specific portal url computation
- Generic URL computation
- In your App
  - access_token definition
  - specific portal_url computation
  - /mail/view controller check and redirection to portal

```python
from odoo.mail.controllers.main import MailController

class OrderController(MailController):

    def _redirect_to_record(self, model, res_id,
acces_token):
        # check access rights and access token
        ...
        if share_user and access_granted:
            return self.portal_url
        return super()
```

The graph


SO SCIENTIFIC
MUCH GRAPH
WOW
MANY LABELS
AMAZING DATA


Complexity
Power
0
1
2
3
'utm.mixin'
'website.published'
'rating.mixin'
'portal.mixin'
'mail.activity.mixin'
'website.seo.metadata'
'mail.thread'

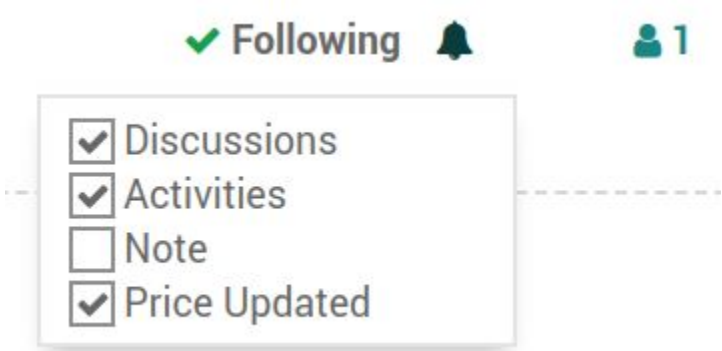
— Still our Classy Cool Dev

Advanced Mail Thread

Mail Thread: Subtypes

- Characterize / filter messages
- Subtype definition (XML)
  - model specific (e.g. plant)
  - default / hidden
- Use in code: message_post

```python
class Plant(models.Model):
    _name = 'plant.plant'
    _inherit = ['mail.thread']

    def write(self, values)::
        res = super()
        if 'price' in values:
            self.message_post('Price Updated',
subtype='plant_price')
        return res
```




```xml
<record id="plant_price" model="mail.message.subtype">
    <field name="name">Price Updated</field>
    <field name="res_model">plant.plant</field>
    <field name="default" eval="True"/>
    <field name="hidden" eval="False"/>
    <field name="description">Price Updated</field>
</record>
```

Mail Thread: Tracking

- Track value change
- track_visibility on field
  - onchange
  - always
- Automatic logging

```python
class Plant(models.Model):
    _name = 'plant.plant'
    _inherit = ['mail.thread']

    price = fields.Float('Price',
        track_visibility='onchange')
```

Mail Thread: Tracking & Subtypes

- Warn on some specific value change
- Link a set of tracking and a subtype
  - _track_subtype
  - return xml_id
- Message with tracking generated with the subtype

  -> allow people to be notified

```python
class Plant(models.Model):
    _name = 'plant.plant'
    _inherit = ['mail.thread', 'rating.mixin']

    price = fields.Float('Price',
        track_visibility='onchange')

    def _track_subtype(self, values):
        if 'price' in values:
            return 'plant_nursery.plant_price'
        return super()
```

Mail Thread: Tracking & Mailing

- Email on some specific value change
- Link a set of tracking and an automatic mailing
  - _track_template
  - give a mail.template and options
- Template rendered and sent
  -> force mailing to people
- Ultra power: add rating in template

```python
class Plant(models.Model):
    _name = 'plant.plant'
    _inherit = ['mail.thread', 'rating.mixin']

    price = fields.Float('Price',
        track_visibility='onchange')

    def _track_template(self, values):
        res = super()
        if 'price' in values:
            res['price'] = (
                'plant_price_template',
                options)
        return res
```

Woody Cutters, Administrator - 3 minutes ago ☆

Hello,

Plant Apple Tree price updated to 50.0.

Email automatically sent by Odoo Plant Nursery for Woody Cutters

Mail Alias

- Email integrated with mailgateway
- Purpose: create / update thread in a given model
- Owner: record defining the alias
- Example
  - alias on category
  - creating XMas quote

```python
class MailAlias(models.AbstractModel):
    _name = 'mail.alias'

    name = fields.Char('Email')

    # record creation / update
    model = fields.Char('Model')
    thread_id = fields.Integer('Update specific record')

    # record owner
    parent_model = fields.Char('Owner model')
    parent_thread_id = fields.Integer('Owner ID')
```

Mail Alias: how to use

- mail.alias.mixin
- add alias_id field
- Specify what do do with alias by overriding methods

```python
class Category(models.Model):
    _name = 'plant.category
    _inherit = ['mail.alias.mixin']

    def get_alias_model_name(self, values):
        return 'plant.order'

    def get_alias_values(self):
        values = super()
        values['alias_defaults'] = {}
        return values
```

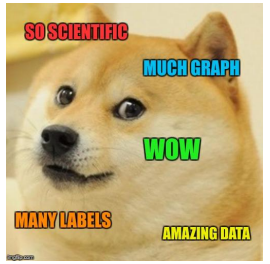
Name	Palm
Alias	tde+xmas@odoo.com

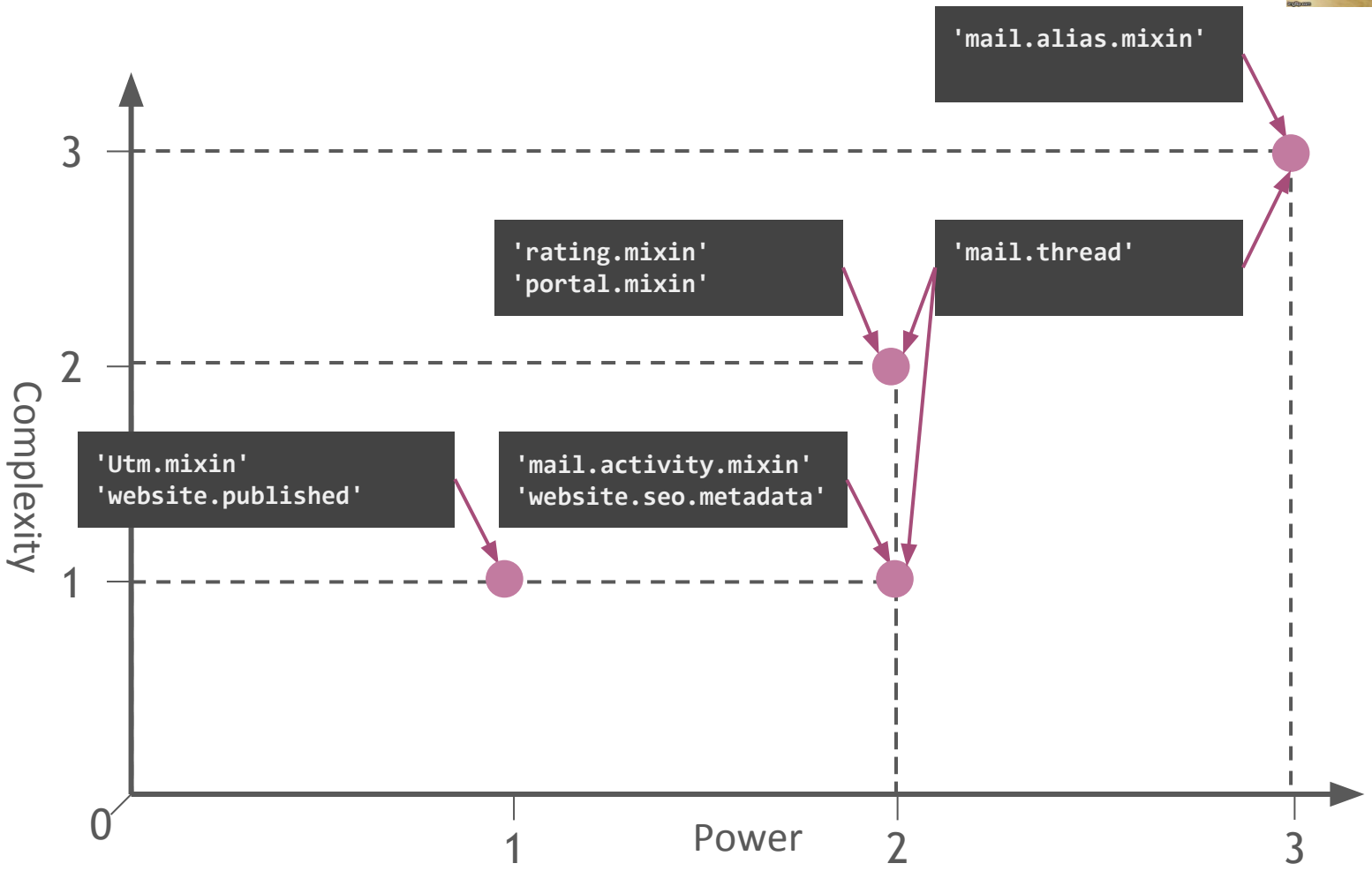
```python
class MailAlias(models.AbstractModel):
    _name = 'mail.alias.mixin'
    _inherit = {'mail.alias' : 'alias_id'}

    alias_id = fields.Many2one('Alias')
```

6

I have a headache

The graph


SO SCIENTIFIC
MUCH GRAPH
WOW
MANY LABELS
AMAZING DATA


'mail.alias.mixin'
'rating.mixin'
'portal.mixin'
'mail.thread'
'Utm.mixin'
'website.published'
'mail.activity.mixin'
'website.seo.metadata'
Complexity
Power
0
1
2
3
1
2
3

2017

odoo

EXPERIENCE

https://www.odoo.com/documentation

https://github.com/tde-banana-odoo/odoo_plants

Thibault DELAVALLEE  • R&D Engineer

based on work from Damien BOUVY

2017

odoo

EXPERIENCE

Thank you.

#odooexperience

Thibault DELAVALLEE  • R&D Engineer