VIEW MODELS

Stop leaking your domain to your views

```php
class Post
{

    public function author(): PostAuthor
    {
        return $this->author;
    }

    public function title(): PostTitle
    {
        return $this->title;
    }

    public function content(): PostContent
    {
        return $this->content;
    }

    public function createdAt(): DateTimeImmutable
    {
        return $this->createdAt;
    }

    public function publishedAt(): ?DateTimeImmutable
    {
        return $this->publishedAt;
    }

    public function isPublished(): bool
    {
        return $this->publishedAt instanceof DateTimeImmutable;
    }

    public function isRecentlyPublished(): bool
    {
        return $this->isPublished() && $this->publishedAt->diff(new
DateTimeImmutable())->days < 10;
    }
}
```

RE-CAP...

- Better Entities
- Entities have business rules
- Entities control state
- Entities control validity

- How about removing getters?

See https://github.com/dave-redfern/better-entities

REMOVE THE GETTERS?!

- K… what? For real?
- How do I read my entity?
- How do I get the state out?
- How do I display to users?

- SHOULD your output use your domain objects?

```php
class Post implements RaisesDomainEventsContract
{

    public static function create(PostAuthor $author, PostTitle
$title, PostContent $content)
    {
    }

    public static function createAndPublish(PostAuthor $author,
PostTitle $title, PostContent $content)
    {
    }

    public function publish(DateTimeImmutable $publishedAt = null)
    {
        $this->publishedAt = ($publishedAt ?: new
DateTimeImmutable());
        $this->updatedAt   = new DateTimeImmutable();
    }

    public function removeFromPublication()
    {
        $this->publishedAt = null;
        $this->updatedAt   = new DateTimeImmutable();
    }

    public function changeTitle(PostTitle $title)
    {
        $this->title     = $title;
        $this->updatedAt = new DateTimeImmutable();
    }

    public function replaceContentWith(PostContent $content)
    {
        $this->content   = $content;
        $this->updatedAt = new DateTimeImmutable();
    }

    public function leaveComment(Commenter $commenter, string
$comment)
    {
        $this->comments->add(new Comment($this, $commenter,
$comment));
        $this->updatedAt = new DateTimeImmutable();
    }
}
```

BENEFITS

- Entities have a single role
- Cannot use them for views
- Can only change state
- Simpler API
- Remember Collections?
  - No problems now!

```php
class EntityAccessor
{

    public static function get($object, $property)
    {
        $refObject = new \ReflectionObject($object);
        $refProp   = $refObject->getProperty($property);
        $refProp->setAccessible(true);

        return $refProp->getValue($object);
    }
}
```

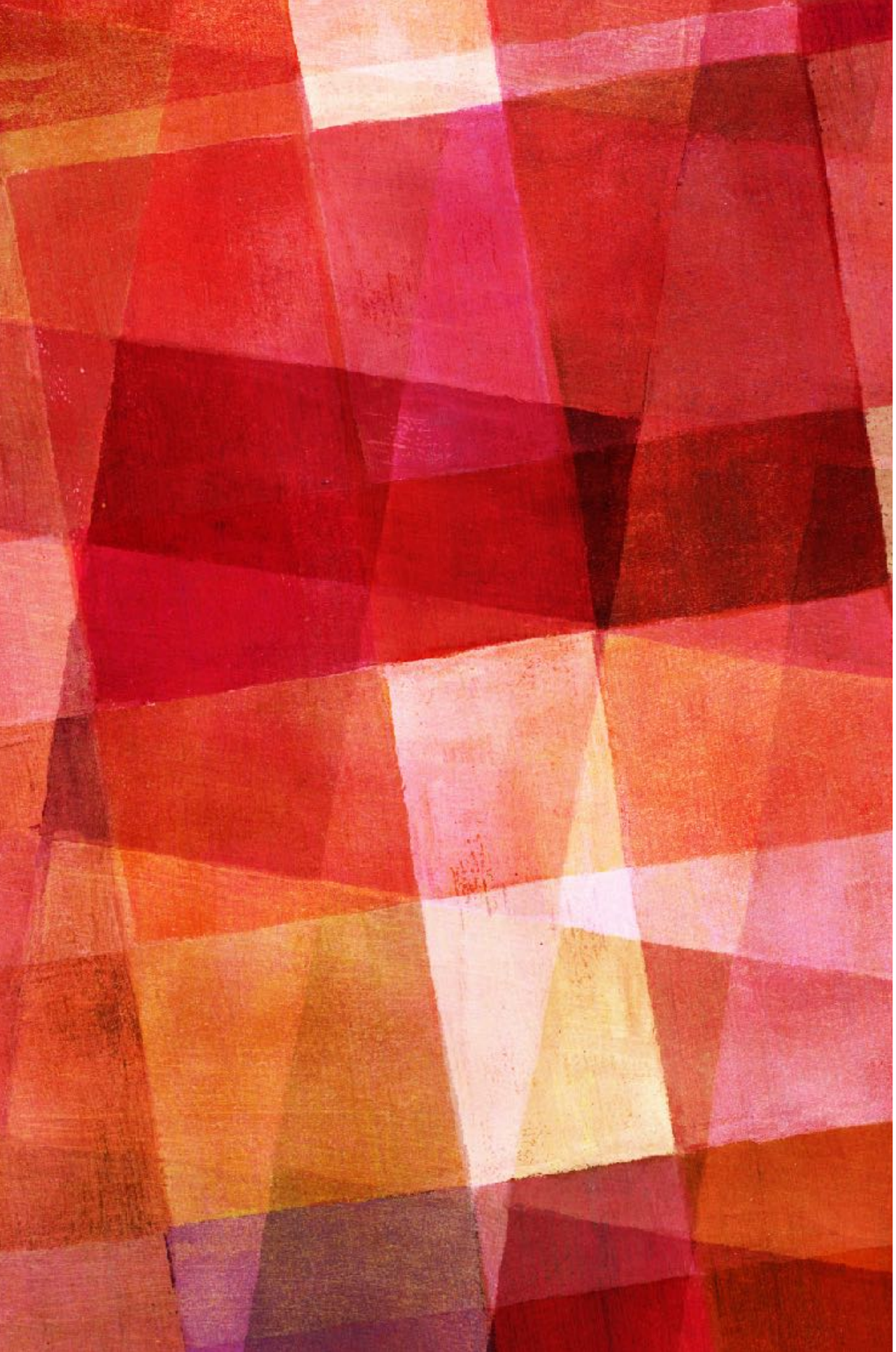
```php
class PostTest extends TestCase
{

    public function testCreate()
    {
        $entity = Post::create(
            $pa = new PostAuthor('bob', new
EmailAddress('bob@e.ca')),
            $pt = new PostTitle('Test Post'),
            $pc = new PostContent('<p>This is a test post</p>')
        );

        $this->assertTrue(
            $pa->equals(
                EntityAccessor::get($entity, 'author')
            )
        );
    }
}
```

DOWNSIDE

- Testing is harder
- Need Reflection
- Still good to test assignment
    - Catches human typos
- Use VOs to check equality
- And my views?

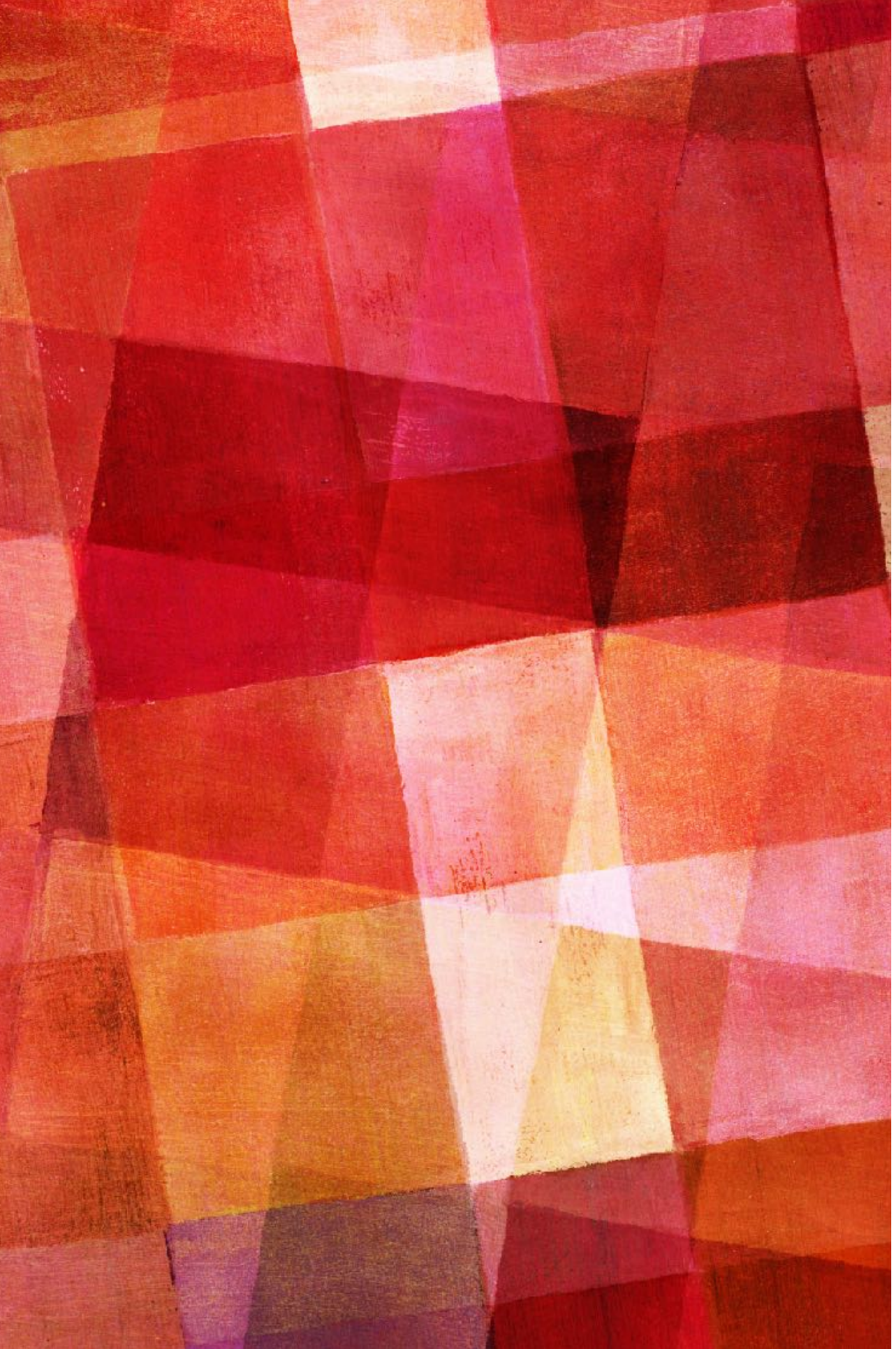
ENTER VIEW MODELS

a.k.a de-coupling your data representation from the domain



VIEW MODELS (VM)

- Provide data API for view(s)
- Read-only
- Encapsulate the VIEWs needs
- Disconnected from domain
- Can be complex
- Can be completely different to entities



VM CONS

- Does duplicate some logic
- Workflow “State” changes must be kept in-sync
- Temptation to share with entities (traits)
  - DON’T do it!
- Temptation for God objects
  - DON’T do it!
  - Be specific

```php
class PostModel
{

    private $slug;
    private $title;
    private $author;
    private $content;
    private $comments;


    public function __construct($slug, $title, $authorName,
$authorEmail, $content)
    {
        $this->slug     = $slug;
        $this->title    = $title;
        $this->author   = new PostAuthor($authorName, new
EmailAddress($authorEmail));
        $this->content  = new PostContent($content);
        $this->comments = new ArrayCollection();
    }


    public function slug(): string
    {
        return $this->slug;
    }

    public function title(): string
    {
        return $this->title;
    }

    public function author(): PostAuthor
    {
        return $this->author;
    }

    public function content(): PostContent
    {
        return $this->content;
    }
}
```

EXAMPLE VM

- Only major Post details
- No ID (using slug)
- Re-using VOs
- Comments collection
- Getters (without get)

```php
class PostModel
{

    private $slug;
    private $title;
    private $author;
    private $content;
    private $comments;


    public function __construct($slug, $title, $authorName,
$authorEmail, $content)
    {
        $this->slug     = $slug;
        $this->title    = $title;
        $this->author   = new PostAuthor($authorName, new
EmailAddress($authorEmail));
        $this->content  = new PostContent($content);
        $this->comments = new ArrayCollection();
    }


    public function slug(): string
    {
        return $this->slug;
    }

    public function title(): string
    {
        return $this->title;
    }

    public function author(): PostAuthor
    {
        return $this->author;
    }

    public function content(): PostContent
    {
        return $this->content;
    }
}
```

PROBLEMS WITH EXAMPLE

- Loads post content
  - Are we even using it?
  - Could be huge
- Do we want comments?
  - Only comment count?
- Published date?
- May need more than one VM

```php
namespace AppBundle\Services\ReadRepositories;

class PostModelRepository
{

    protected $posts;

    public function __construct(PostRepository $posts)
    {
        $this->posts = $posts;
    }

    public function findLatestPosts($limit = 10)
    {
        $qb = $this->posts->createQueryBuilder('p');
        $qb
            ->select(sprintf(
                'NEW %s(
                    p.slug, p.title, p.author.name, p.author.email,
                    p.content.content
                 )',
                    PostModel::class
                )
            )

            ->where('p.publishedAt <= :now')
            ->orderBy('p.publishedAt', 'DESC')
            ->setMaxResults($limit)
            ->setParameter(':now', new \DateTime('-14 days'))
        ;

        return $qb->getQuery()->getResult();
    }
}
```

FETCHING THE VM

- Use a custom Repository
- Use PostRepository for QB
- DQL to fetch a “NEW” object
- Returns array of PostModels

- Coupled to entity structure
- Relies on Doctrine

```php
class PostModelPdoRepository
{

    protected $conn;

    public function __construct(Connection $conn)
    {
        $this->conn = $conn;
    }

    public function findLatestPosts($limit = 10)
    {
        $query = '
            SELECT p.title_slug, p.title_title, p.author_name,
                   p.author_email, p.content_content
              FROM posts p
             WHERE p.published_at <= :now
             ORDER BY p.published_at DESC
        ';

        $stmt = $this->conn->prepare($query);

        $stmt->setFetchMode(PDO::FETCH_CLASS, PostModel::class);

        $stmt->execute([
            ':now' => (new \DateTime('-14 days'))
                        ->format('Y-m-d H:i:s')
        ]);

        return $stmt->fetchAll();
    }
}
```

NO DQL JUST PDO

- Can fetch into a Class
- No DQL needed - pure SQL
- Returns an array of PostModels

- Watch out!
  - Constructor calling buggy
  - Selected fields become properties (naming)
  - PROPS_LATE sets props after constructor

```php
class PostModel
{

    public function comments(): ArrayCollection
    {
        return $this->comments;
    }

    public function attachComments(ArrayCollection $comments)
    {
        $this->comments = $comments;
    }
}
```

```php
class PostModelRepository
{
    protected function loadPostWithComments(PostModel $post)
    {
        $qb = $this->posts->createQueryBuilder('p');
        $qb
            ->select(sprintf(
            NEW %s(c.commenter.name, c.commenter.email, c.comments)',
            CommentModel::class
            ))
            ->innerJoin('p.comments', 'c')
            ->where('p.title.slug = :slug')
            ->orderBy('c.createdAt', 'ASC')
            ->setParameter(':slug', $post->slug())
        ;

        $post->attachComments(
            new ArrayCollection($qb->getQuery()->getResult())
        );

        return $post;
    }
}
```

HOW TO GET THE COMMENTS?

- Separate method
- Load comments as needed
- Load from slug on Post
- Add method to set comments

- Limit comments?
- Paginate?
    - Easy to do!

```
AppBundle\ViewModels\PostModel {#1213
  -slug: "test-post"
  -title: "Test Post"
  -author: AppBundle\Entities\ValueObjects\PostAuthor {#1227
    -name: "bob"
    -email: AppBundle\Entities\ValueObjects\EmailAddress {#1231
      -email: "bob@example.com"
    }
  }
  -content: AppBundle\Entities\ValueObjects\PostContent {#1212
    -content: "<p>This is a test post</p>"
  }
  -comments: Doctrine\Common\Collections\ArrayCollection {#1217
    -elements: array:4 [
      0 => AppBundle\ViewModels\CommentModel {#1250
        -commenter: AppBundle\Entities\ValueObjects\Commenter {#1246
          -name: "Bob"
          -email: AppBundle\Entities\ValueObjects\EmailAddress {#1253
            -email: "bob@example.com"
          }
        }
        -comment: "These are some comments."
        -date: DateTimeImmutable {#1221
          +"date": "2017-06-25 23:23:46.000000"
          +"timezone_type": 3
          +"timezone": "UTC"
        }
      }
      …
    ]
  }
}
```

TEST THE RESULTS

- Add functional tests
- Persists via Entities
- Fetch as PostModels
- VOs were instantiated
- Comments loaded + VOs

See *ViewModelIntegrationTest.php* in sample code

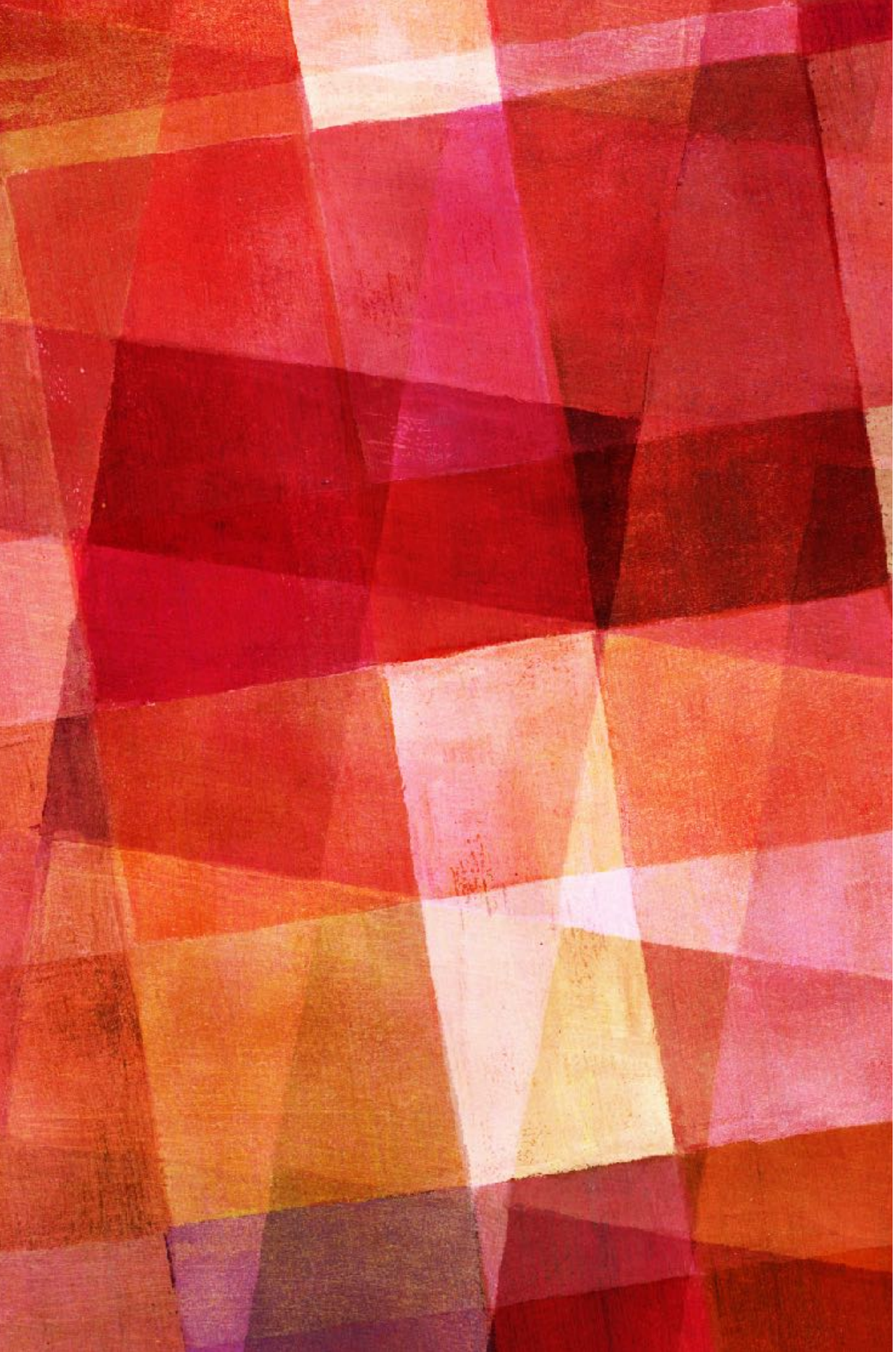
GOING FURTHER

Custom helper methods

```php
class PostModel
{
    public function authorLink()
    {
        return sprintf('<a href="mailto:%s?subject=Re:%s">%s</a>',
            $this->author->email(),
            urlencode($this->title),
            $this->author->name()
        );
    }

    public function postLink($route)
    {
        return sprintf('<a href="%s/%s">%s</a>',
            $route, $this->slug, $this->title
        );
    }

    public function reverseCommentOrder()
    {
        return new ArrayCollection(
            array_reverse($this->comments->toArray())
        );
    }

    public function findCommentsBy(Commenter $commenter)
    {
        return $this->comments->filter(
            function ($comment) use ($commenter) {
                /** @var CommentModel $comment */
                return $comment->commenter()->equals($commenter);
            }
        );
    }

    public function findCommentsContaining(string $keyword)
    {
        return $this->comments->filter(
            function ($comment) use ($keyword) {
                /** @var CommentModel $comment */
                return $comment->contains($keyword);
            }
        );
    }
}
```

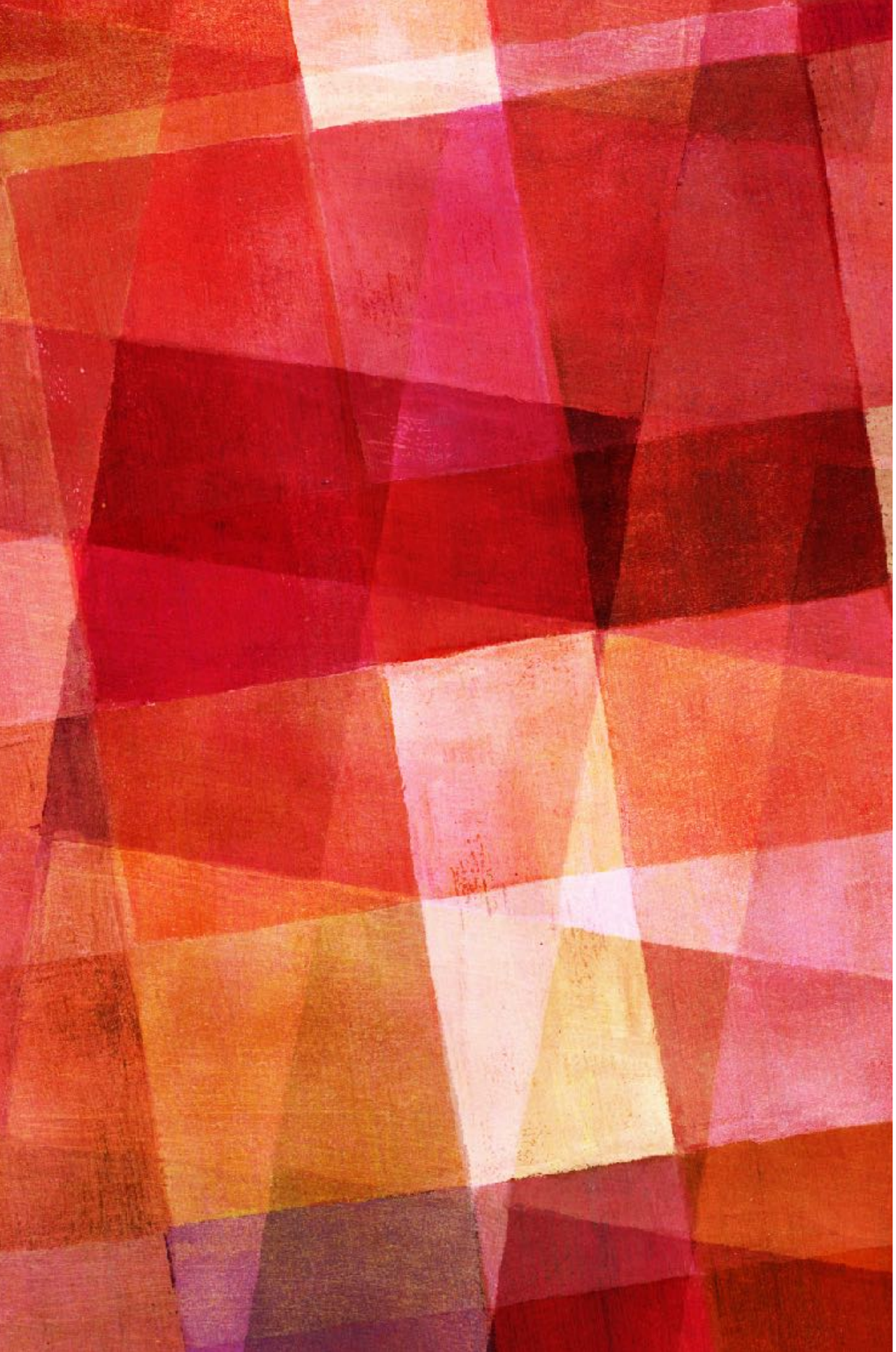
MODEL HELPERS

- Embed linking methods
  - Link to
  - Call to
  - Contact Author etc.
- Icons on state
- Collection helpers



A MODEL FOR ALL OCCASIONS

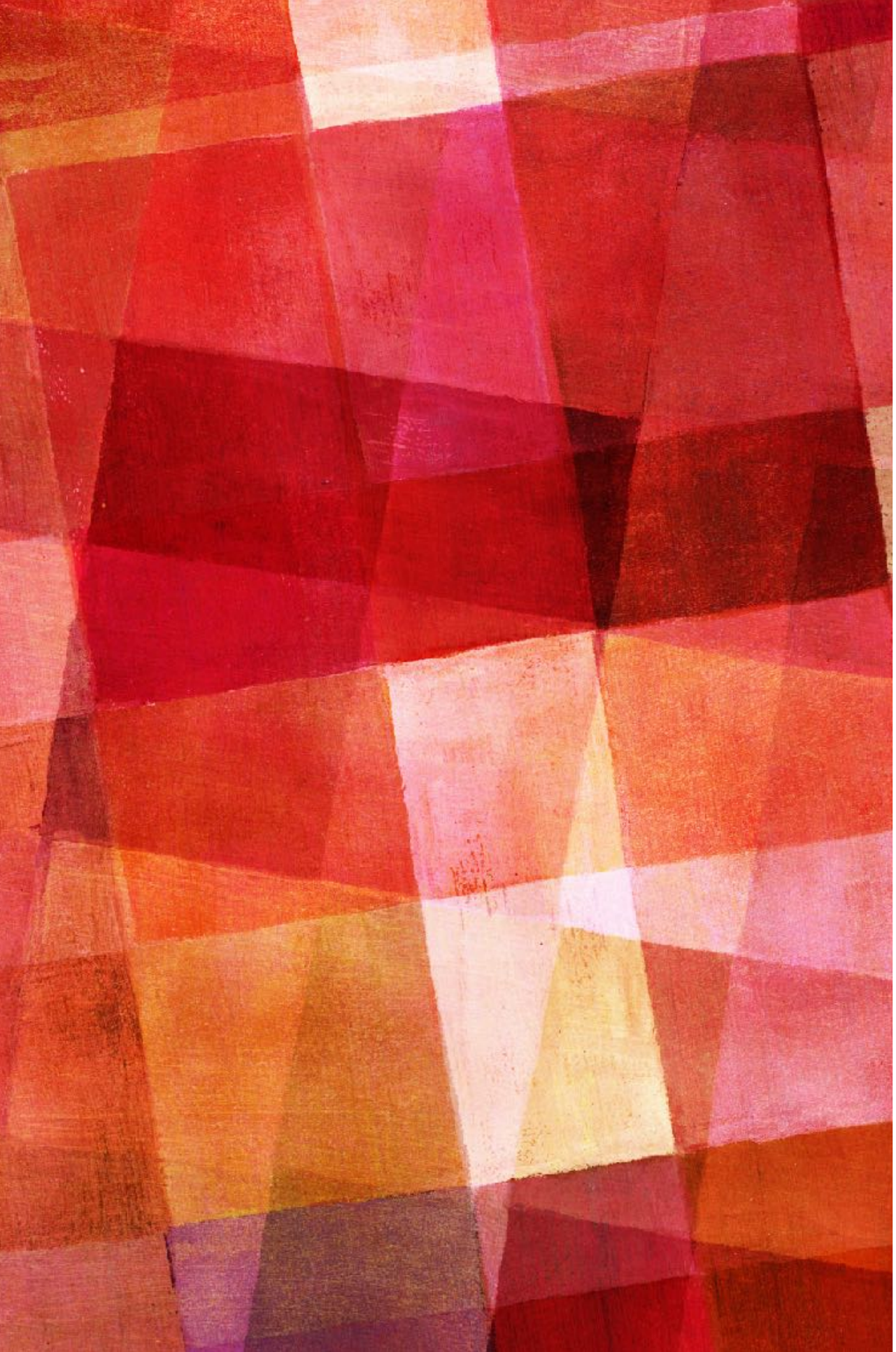
- Reporting
  - Use custom models
  - All data is read-only
  - Keep domain out
  - Maintain context
  - Use report language
- Do you even need a model?



CQRS

- Design pattern
- Separate changing / reading
  - Commands change
  - Queries read
- Commands use entities
- Queries return View Models

- Related: Event Sourcing



MORE...

- Command Query Responsibility Separation
- Event Sourcing
- Revisit "Better Entities”
  - https://github.com/dave-redfern/better-entities
- Example code available
  - https://github.com/dave-redfern/view-models