PHP DEV Standards

Table of Contents

Design patterns

Dependency injection

https://en.wikipedia.org/wiki/Dependency_injection

You should avoid direct dependencies.

Any dependency on other objects should be via __construct() or as function parameter

You should not use keyword new

Only exception can be factories (Where object is created and returned)

```
function doingSomethingWrong(int $number)
{
    $object = new MyClass();
    $object->number = $number;


    // do stuff
}

function doingSomethingBetter(MyClass $object)
{
    // Do stuff
}
```

Slateless

Do not save any state (cache is not a state)

Class will get needed dependencies (factories, repositories, resolvers ...) in __construct()

All other parameters (for example: for decision making) will get as function parameter.

Nothing will be saved, only returned

> **Don't**
>
> Do not write setters, if not neccessary

```
final class myResolver
{
   private $myValidator;


   public function __construct(MyValidator $myValidator)
   {
     $this->myValidator = $myValidator;
   }


   public function resolve(MyRequest $request) : ResolveObject
   {
      if ($this->myValidator->isValid($request) === false) {

        // Fail fast
        throw new MyException("Very usefull message");
      }


     // Do resolve stuf
     // ...
     // ...


     return $myResolveObject;
   }


   public function setValidator(MyValidator $myValidator) : void
   {
          // This is fundamentaly bad. Don't do anything like this
   }


}
```

Null is evil

www.google.cz/search?client=opera&hs=lb7&ei=mGtdXJnrD4iclwTU05bIAQ&q=null+is+evil

Do not return null. There is always better way to handle non-initialized variables.

For example, when returning an array with no items, return empty array and not null

```
function getArray(bool $returnArray): ?array
{
    if (rand(1, 11) % 2 === 0) {
        return [1];
    }

    return null;
}

function getArrayBetter(bool $returnArray): array
{
    if (rand(1, 11) % 2 === 0) {
        return [1];
    }

    return [];
}
```

Global - don't

Global state/space is good to left intact.

Global (static) function often creates tight dependencies - it is hard to test, it is hard to change, code using static functions is not 100% re-usable

```
# Do not

class MyCalculator
{
   public static function myAwesomeCalculus() : int
   {
     return 4;
   }
}

Class MainClass
{
   public function calculate() : void
   {
     ////
     $variable = MyCalculator::myAwesomeCalculus();
     ////
   }
}
```

```
# Do

class MyCalculator implements MyCalculatorInterface
{
   public function myAwesomeCalculus() : int
   {
     return 4;
   }
}


Class MainClass
{
   public function calculate(MyCalculatorInterface $calculator) : void
// See this awesome, interface based dependency injection
   {
     ////
     $variable = $calculator->myAwesomeCalculus();
     ////
   }
}
```

Configuration switch

Use only positive names

Configuration switch

```
<?php (or Yaml)


// Good name
const WSC_ENABLED = true;

// Bad name - Do not use it
const WSC_DISABLED = false;
```

Early return

Using early return is recommended when it improves readability and overall code understanding, speed and aesthetics

- It reduces indention later on

```
<?php


function withIndention (MyObject $item): void
{
        if ($item->a === 'A') {
       if ($item->b === 'B') {
                        doStuff();
        }
    }
}


// Early return reduces indention and improve readability
function withoutIndention (MyObject $item): void
{
        if ($item->a !== 'A') {
                return;
    }


        if ($item->b !== 'B') {
                return;
    }


        doStuff();
}
```

- Improves speed

```
function speedyFunction (ArrayObject $array, int
findingThisNumber): ArrayItem
{
   foreach ($array as $item) {
     if ($item->number === $findingThisNumber) {
        return $item;
       }
   }


        throw new \Exception("This should never happen");
}
```

Do not forget to maintain consistency, within function/class scope

```
// Try your best to avoid mixing early and non-early return within the
function
function mixedFunction (int $a, int $b): int
{
  $c = 0;


  if ($a === 1) {
    return 0;
  }


  if ($a === 2 ) {
    $c = 2;


    if ($b === 3) {
      $c += $b;


      if ($a === 4) {
        return $a + $b;
      }

    }

   $c += 5;


   }

        return $c;
}
```

PHP 7+

Use typing in php at maximum

As function parameters, as return types, everywhere

```
declare(strict_types=1);

function sum(int $a, int $b) : int
{
    return $a + $b;
}
```

Exception

Exceptions should be clearly named

Do not use generic \Exception

Do not make logic based on exceptions

```
function makeBadException(): void
{
  throw Exception("Random string without code");
}


function goodException(): void
{
  throw SpecificException("Specific message or even a code");
}
```

Constants

Every string or number should be externalized in constants

```
function magicInteger(int $coffee): int
{
  return  12 * $coffee; // What is 12 ?
}


function notSoInteger(int $coffee): int
{
  return COFFEE_PER_DAY * $coffee; // Clearly described constant; Apply
same logic to strings/texts
}
```

Annotation

- Write annotation only for added value.
- Write annotation for specifying array.
- Write annotation for exceptions.

- When writing an annotation, write full one.

```
<?php

class GoodAnnotation
{

   public function getMeCoolObject(int $a, int $b) : CoolObject
   {
     // ...
   }

         /**
         * @param int $a
    * @param int $b
    * @return array|CoolObject[]
        * @throw MyBadException
    */
    public function getMeArrayOfCoolObjects(int $a, int $b): array
    {
       // ...
           throw new MyBadExecption ('numbers $a and $b holds same
value');
    }
}
```

Unit tests

All code, which is not integrating something (database, other system), should be under unit tests

> Every logic path should be under test!

Not easily testable function/class

```
class NotEasilyTestAble
{
    // Database repository or other
    private $coolObjectRepository;

    // A lot of is going inside this function
    public function dothings(int $number, string $word) : int
    {
        // Integration dependency (database, other systems...)
        $coolObject = $this->coolObjectRepository->findById($number);

        // Some logic
        $sentence = $coolObject->getWords() . $word;
        $sentenceLength = strlen($sentence);

        if ($sentenceLength < MAXIMUM_ALLOWED_SENTENCE_SIZE) {
            throw SpecificException("Specific expension");
        }

        return $sentenceLength;
    }
}
```

Easily testable

```
class EasilyTestAble
{
    // Database repository or other
    private $coolObjectRepository;

    // Resolver for logic
    private $sentenceLengthResolver;

    public function dothings(int $number, string $word) : int
    {
        // Integration dependency (database, other systems...)
        $coolObject = $this->coolObjectRepository->findById($number);

        // Some logic - extracted to its own class
        $sentenceLength = $this->sentenceLengthResolver-
>resolveFromCoolObjectAndWord($coolObject, $word);

        return $sentenceLength;
    }
}

// This resolver is usable in other classes too!
class SentenceLengthResolver
{
    // Easily testable, reusable function
    // This class (and function) has only one specific purpose!
    public function resolveFromCoolObjectAndWord (CoolObject
$coolObject, string $word) : string
    {
        $sentence = $coolObject->getWords() . $word;
        $sentenceLength = strlen($sentence);

        if ($sentenceLength < MAXIMUM_ALLOWED_SENTENCE_SIZE) {
            throw SpecificException("Specific expension");
        }

        return $sentenceLength;
    }
}
```

PHP PSR

Code is compliance with PSR2 https://www.php-fig.org/psr/psr-2/

Code complexity

Its good to maintain low code complexity

5 minutes read here: https://modess.io/npath-complexity-cyclomatic-complexity-explained/

```
// NPath complexity 4 - 4 different paths
function foo($a, $b)
{
    if ($a > 10) {
        echo 1;
    } else {
        echo 2;
    }
    if ($a > $b) {
        echo 3;
    } else {
        echo 4;
    }
}
```

Code Review

What to look for:

- Is the code doing what it should ? - Look into Jira story
- Is the business logic and integration separated ?
- Are all (business logic) paths covered in unit tests ?
- Build is green
- Are new/changed/deleted parameters in release notes ?
- Is the code fulfilling DEV standard covered on this page ?
- Is the code easily readable ?
- (Deeper knowledge) Is the code safe ? - Does not break any other app

Static code analysis

PHP STAN

PHP STAN - https://github.com/phpstan/phpstan

All new project should comply with PHP STAN LEVEL 7

PHP CodeSniffer

PHP CS - https://github.com/squizlabs/PHP_CodeSniffer

Online dev package:

https://bitbucket.oskarmobil.cz/projects/ONLINE/repos/coding-standards/browse

Other more generic design patterns

Factory pattern

https://en.wikipedia.org/wiki/Factory_method_pattern#PHP

Naming

All classes, folders/namespaces shall have good and reasonable name

General is good to be compliance with https://www.php-fig.org/psr/psr-2/

<<< VOTING >>> - some stuff here is subject to voting

Command

You shall use command for console commands

Command example

```
<?php declare(strict_types=1);

namespace WSCBE\Command;

use Exception;
use Symfony\Component\Console\Input\InputInterface;
use Symfony\Component\Console\Output\OutputInterface;
use Vodafone\LoggerBundle\Monolog\Logger;
use WSCBE\Service\ChatApi\Facade\MessengerNotificationWorker;
use WSCBE\Service\Maintenance\MaintenanceService;

class MessengerNotificationCommand extends AbstractCommand
{
    private $messengerNotificationWorker;

    public function __construct(
        Logger $logger,
        MaintenanceService $maintenanceService,
        MessengerNotificationWorker $messengerNotificationWorker
    ) {
        $this->messengerNotificationWorker =
$messengerNotificationWorker;
        parent::__construct($logger, $maintenanceService);
    }

    protected function configure()
    {
        $this->setName('wscbe:messenger:deleteOutdatedNotifications')
            ->setDescription($this->getCommandIdentifier() . ' cron for
delete outdated.');
    }

    protected function executeInternal(InputInterface $input,
OutputInterface $output)
    {
        $this->logger->addInfo($this->getCommandIdentifier() . ' -
```

```
DeleteOutdated: start');

        try {
            $this->messengerNotificationWorker-
>deleteOutdatedNotifications();

            // Log ending
            $this->logger->addInfo($this->getCommandIdentifier() . ' -
DeleteOutdated: ended successfully');
        } catch (Exception $e) {
            $this->logger->addError(
                $this->getCommandIdentifier() . ' - DeleteOutdated:
process failed with unhandled exception',
                [
                    'exception' => get_class($e),
                    'message' => $e->getMessage()
                ]
            );

            return CommandReturnCodeEnum::COMMAND_FAILED;
        }

        return CommandReturnCodeEnum::COMMAND_SUCCESSFUL;
    }

    protected function getCommandIdentifier()
    {
        return 'Messenger notification';
    }
}
```

Entity

Entity is used for describing database entity/table or rest/soap request/response

Entity example

```
<?php declare(strict_types=1);

namespace WSCBE\Entity\Attachments;

use DateTime;
use Doctrine\ORM\Mapping as ORM;
use WSCBE\Component\OracleConnector\Doctrine\Annotation as
WscBeAnnotation;
use WSCBE\Component\Utility\StringsTrait;
use WSCBE\Entity\Eligibility\RoleList;
use WSCBE\Service\Attachment\AttachmentStatusEnum;
```

```
/**
 * Class AttachmentToDmsViaBssEntity
 *
 * @ORM\Table(
 * name="ATTACHMENT_BSS_DMS_QUEUE",
 * indexes={
 * @ORM\Index(name="ATTACHMENT_BSS_DMS_IX_FIND", columns={"STATUS",
"ERROR_COUNT"})
 * }
 * )
 * @ORM\Entity(repositoryClass="
WSCBE\Repository\Attachments\AttachmentToDmsViaBssRepository")
 * @WscBeAnnotation\Tablespace(type="data", size="big")
 */
class AttachmentToDmsViaBssEntity
{
 use StringsTrait;

 /**
 * @var integer
 * @ORM\Column(name="ID", type="integer")
 * @ORM\Id
 * @ORM\GeneratedValue(strategy="SEQUENCE")
 * @ORM\SequenceGenerator(sequenceName="
ATTACHMENT_BSS_DMS_QUEUE_ID_se", allocationSize=1, initialValue=1)
 */
 private $id;

 /**
 * @var string like incident ID
 * @ORM\Column(name="ROOT_ID", type="string", length=100)
 */
 private $rootId;

 /**
 * @var string|null
 * @ORM\Column(name="AAG_ID", type="string", nullable=true)
 */
 private $aagId;

 /**
 * @var integer|null like comment ID
 * @ORM\Column(name="PARENT_ID", type="integer", nullable=true)
 */
 private $parentId;

 /**
 * @var string
 * @ORM\Column(name="FILENAME", type="string", length=250)
 */
 private $filename;
```

```
 /**
 * @var string
 * @ORM\Column(name="CONTENT_TYPE", type="string", length=250)
 */
 private $contentType;

 /**
 * @var resource
 * @ORM\Column(name="CONTENT", type="blob")
 */
 private $content;

 /**
 * @var string
 * @ORM\Column(name="UPLOAD_TYPE", type="string", length=250,
nullable=false)
 */
 private $uploadType;

 /**
 * @var string $status
 * @ORM\Column(name="STATUS", type="string", options={"default" :
"new"})
 */
 private $status;

 /**
 * @var int $errorCount
 * @ORM\Column(name="ERROR_COUNT", type="integer", options={"default" :
0})
 */
 private $errorCount;

 /**
 * @var string
 *
 * @ORM\Column(name="MSISDN", type="string", length=16, nullable=true)
 */
 private $msisdn;

 /**
 * @var string|null
 * @ORM\Column(name="ERROR_MSG", type="text", nullable=true)
 */
 private $errormsg;

 /**
 * @var DateTime
 *
 * @ORM\Column(name="CREATED_AT", type="datetime", nullable=false)
 */
 private $createdAt;
```

```
 /**
 * @var DateTime|null
 *
 * @ORM\Column(name="LAST_PROCESSED", type="datetime", nullable=true)
 */
 private $lastProcessed;

 /**
 * @var string|null
 * @ORM\Column(name="PRIVATE_LINK", type="text", nullable=true)
 */
 private $privateLink;

 /**
 * @var string|null
 * @ORM\Column(name="PUBLIC_LINK", type="text", nullable=true)
 */
 private $publicLink;

 /**
 * @var RoleList|null
 * @ORM\ManyToOne(targetEntity="WSCBE\Entity\Eligibility\RoleList")
 * @ORM\JoinColumn(name="ROLE_ID", referencedColumnName="ID",
nullable=true)
 */
 private $authLevel;

 /**
 * @var string|null
 * @ORM\Column(type="string", length=20, nullable=true)
 */
 private $caNumber;

 /**
 * @ORM\Column(type="string", nullable=true)
 * @var string|null
 */
 private $nameId;

 public function __construct(
 string $rootId,
 ?string $aagId,
 string $filename,
 string $contentType,
 string $content,
 string $uploadType,
 ?string $msisdn,
 ?RoleList $authLevel,
 ?string $caNumber = null,
 ?DateTime $createdAt = null
 ) {
 $this->rootId = $rootId;
 $this->aagId = $aagId;
```

```
 $this->filename = $filename;
 $this->contentType = $contentType;
 $this->content = $content;
 $this->uploadType = $uploadType;
 $this->msisdn = $msisdn;
 $this->authLevel = $authLevel;
 $this->errorCount = 0;
 $this->status = AttachmentStatusEnum::NEW;
 $this->caNumber = $caNumber;

 if ($createdAt === null) {
 $this->createdAt = new DateTime;
 }
 }
}
```

Enum

Enum is used for enumeration

Enum example

```
<?php declare(strict_types=1);

namespace WSCBE\Service\Attachment;

use Enum\AbstractEnum;

class AttachmentStatusEnum extends AbstractEnum
{
    public const NEW = 'new';
    public const ANTIVIR_CHECK = 'antivir-check';
    public const ERROR_ANTIVIR_CHECK = 'error-antivir-check';
    public const ERROR_BSS = 'error-bss';
    public const ERROR_AARON = 'error-aaron';
    public const SUCCESS = 'success';
}
```

Facade

Is used for Facade design pattern https://github.com/domnikl/DesignPatternsPHP/tree/master/Structural/Facade

Factory

Is good name for a class that is creating other classes

Factory example

```php
<?php declare(strict_types=1);

namespace WSCBE\Service\EmailCare\Factory;

use WSCBE\Entity\Attachments\AttachmentToDmsViaBssEntity;
use WSCBE\Entity\Eligibility\RoleList;
use WSCBE\Service\Attachment\UploadTypeEnum;
use WSCBE\Service\ChatApi\Dto\AttachmentDto;
use WSCBE\Service\EmailCare\Resolver\AttachmentContentTypeResolver;

final class AttachmentEntityFactory
{

    private $contentTypeResolver;

    public function __construct(AttachmentContentTypeResolver
$contentTypeResolver)
    {
        $this->contentTypeResolver = $contentTypeResolver;
    }

    public function create(AttachmentDto $attachment, string
$incidentId, ?RoleList $roleList, ?string $msisdn, ?string
$customerAccountNumber): AttachmentToDmsViaBssEntity
    {
        $contentType = $this->contentTypeResolver->resolve($attachment);

        return new AttachmentToDmsViaBssEntity(
            $incidentId,
            $attachment->getAagId(),
            $attachment->getFilename(),
            $contentType,
            $attachment->getContent(),
            UploadTypeEnum::BSS_CHAT,
            $msisdn,
            $roleList,
            $customerAccountNumber
        );
    }
}
```

Handler

Is used for handling server requests

Handler example

```php
<?php declare(strict_types=1);
```

```
namespace WSCBE\Bundle\SoapServerBundle\Service\Handlers\ChatApi;

use Exception;
use Vodafone\WscshopEntity\Enum\ChatApi\AddIncidentErrorEnum;
use Vodafone\WscshopEntity\Soap\ChatApi\Request\AddIncidentRequest;
use Vodafone\WscshopEntity\Soap\ChatApi\Response\AddIncidentResponse;
use Vodafone\WscshopEntity\Soap\SoapHeaderEntity;
use Vodafone\WscshopEntity\Soap\SoapRequestBaseEntity;
use Vodafone\WscshopEntity\Soap\SoapResponseBaseEntity;
use Vodafone\WscshopEntity\Soap\SoapResponseMessageEntity;
use Vodafone\WscshopEntity\Soap\SoapResponseResultEntity;
use WSCBE\Bundle\SoapServerBundle\Exception\InvalidArgumentException;
use WSCBE\Bundle\SoapServerBundle\Infrastructure\HandlerBase;
use WSCBE\Service\ChatApi\Exception\ContactOnSpamListException;
use WSCBE\Service\ChatApi\Facade\AddIncidentFacade;
use WSCBE\Service\ChatApi\Validator\AddIncidentRequestValidator;

class AddIncidentHandler extends HandlerBase
{
    private $addIncidentFacade;
    private $addIncidentRequestValidator;

    public function __construct(
        AddIncidentFacade $addIncidentFacade,
        AddIncidentRequestValidator $addIncidentRequestValidator
    ) {
        $this->addIncidentFacade = $addIncidentFacade;
        $this->addIncidentRequestValidator =
$addIncidentRequestValidator;
    }

    public function canHandle(SoapRequestBaseEntity $request)
    {
        return $request instanceof AddIncidentRequest;
    }

    /**
     * @param SoapRequestBaseEntity|AddIncidentRequest $request
     * @param SoapHeaderEntity $soapHeaderEntity
     * @return SoapResponseBaseEntity
     * @throws InvalidArgumentException|Exception
     */
    protected function handleInternal(SoapRequestBaseEntity $request,
SoapHeaderEntity $soapHeaderEntity)
    {
        $validationResult = $this->addIncidentRequestValidator-
>validateRequest($request);

        if ($validationResult->isValid() === false) {
            throw new InvalidArgumentException($validationResult-
>getValidationErrorsAsStringArray());
        }
```

```
        try {
            $incident = $this->addIncidentFacade->addIncident(
                $soapHeaderEntity->SessionId,
                $request
            );

            return new AddIncidentResponse(
                $incident,
                SoapResponseResultEntity::getOkResult()
            );

        } catch (ContactOnSpamListException $e) {

            return new AddIncidentResponse(
                null,
                SoapResponseResultEntity::getErrorResult([
                    new SoapResponseMessageEntity(AddIncidentErrorEnum::
CONTACT_ON_SPAM_LIST),
                ])
            );
        }
    }
}
```

Mapper

Is used when you are mapping one object to (something) other

Mapper example

```php
<?php declare(strict_types=1);

namespace WSCBE\Service\EmailCare\Mapper;

use InvalidArgumentException;
use Vodafone\WscshopEntity\Enum\AuthorizationRoleEnum;
use WSCBE\Service\EmailCare\Enum\MFAuthorizationEnum;

final class MFAuthRoleMapper
{
    private const MAP = [
        MFAuthorizationEnum::CA => AuthorizationRoleEnum::ADMIN,
        MFAuthorizationEnum::BA => AuthorizationRoleEnum::BILLING_ADMIN,
        MFAuthorizationEnum::EU => AuthorizationRoleEnum::USER,
    ];

    public function map(string $name): string
    {
        if (!array_key_exists($name, self::MAP)) {
            throw new InvalidArgumentException(sprintf('Role not found
in map: %s', $name));
        }

        return self::MAP[$name];
    }
}
```

Resolver

Can be used to resolve something. For example you want to resolve, if the returned call was success or failure.

Resolve more complex logic

Resolver example

```php
<?php declare(strict_types=1);

namespace WSCBE\Service\ChatApi\Resolver;

use Exception;
use Vodafone\WscshopEntity\Enum\ChatApi\ConversationEventTypeEnum;

class EventTypeResolver
{
    /**
     * @param string $eventType
```

```
     * @return string
     * @throws Exception
     */
    public function resolveEventType($eventType)
    {
        // @refactorMe return ENUM
        // @refactorMe maybe use something like in_array($eventType,
$this->getAgentArrayTypes()....
        switch ($eventType) {
            case ConversationEventTypeEnum::AGENT_POST:
            case ConversationEventTypeEnum::AGENT_ENTERED:
            case ConversationEventTypeEnum::AGENT_PRIVATE_POST:
            case ConversationEventTypeEnum::PRIVATE_NOTE:
            case ConversationEventTypeEnum::FINISHED_STATUS_CHANGED:
            case ConversationEventTypeEnum::AGENT_I_STATUS_CHANGED:
                return "AGENT";
            case ConversationEventTypeEnum::END_USER_POST:
            case ConversationEventTypeEnum::END_USER_I_STATUS_CHANGED:
                return "END_USER";
            case ConversationEventTypeEnum::END_USER_ABSENT:
            case ConversationEventTypeEnum::END_USER_PRESENT:
            case ConversationEventTypeEnum::ENGAGEMENT_INVITATION:
            case ConversationEventTypeEnum::ENGAGEMENT_ACCEPTED:
            case ConversationEventTypeEnum::ENGAGEMENT_DECLINED:
            case ConversationEventTypeEnum::ENGAGEMENT_TIMEOUTED:
            case ConversationEventTypeEnum::TRANSFER_ACCEPTED:
            case ConversationEventTypeEnum::TRANSFER_CANCELED:
            case ConversationEventTypeEnum::TRANSFER_DECLINED:
            case ConversationEventTypeEnum::TRANSFER_INVITATION:
            case ConversationEventTypeEnum::TRANSFER_TIMEOUTED:
            case ConversationEventTypeEnum::TRANSFER_TO_QUEUE:
            case ConversationEventTypeEnum::TRANSFER_VIEW:
            case ConversationEventTypeEnum::CONFERENCE_ACCEPTED:
            case ConversationEventTypeEnum::CONFERENCE_DECLINED:
            case ConversationEventTypeEnum::CONFERENCE_INVITATION:
            case ConversationEventTypeEnum::CONFERENCE_TIMEOUTED:
            case ConversationEventTypeEnum::AGENT_LEAVE:
            case ConversationEventTypeEnum::AGENT_CONCLUDED:
            case ConversationEventTypeEnum::AGENT_DISCONNECTED:
            case ConversationEventTypeEnum::END_USER_LOST:
            case ConversationEventTypeEnum::END_USER_DISCONNECTED:
            case ConversationEventTypeEnum::END_USER_CONCLUDED:
            case ConversationEventTypeEnum::IDLE_TIMEOUT:
            case ConversationEventTypeEnum::MONITOR_BEGIN:
            case ConversationEventTypeEnum::MONITOR_END:
            case ConversationEventTypeEnum::AGENT_ROLE_CHANGED:
            case ConversationEventTypeEnum::QUEUE_TIMEOUT:
            case ConversationEventTypeEnum::END_USER_TYPING_START:
            case ConversationEventTypeEnum::END_USER_TYPING_STOP:
            case ConversationEventTypeEnum::AGENT_TYPING_START:
            case ConversationEventTypeEnum::AGENT_TYPING_STOP:
            case ConversationEventTypeEnum::I_STATUS_CHANGED:
            case ConversationEventTypeEnum::AUTH_LEVEL_CHANGED:
```

```
            case ConversationEventTypeEnum::OTHER:
                return "SYSTEM";
        }

        throw new Exception('Unresolved event type from conversation: '
. $eventType);
    }
}
```

Listener

Use listener, when listening for an event

Service

When you don't know how to name something. Name it service.

Service is usually orchestrating more complex logic.

Service example

```
<?php declare(strict_types=1);

namespace WSCBE\Bundle\RestServerBundle\Service\Payment;

use Vodafone\WscshopEntity\Rest\Payment\Request\PayByCardRequest;
use
WSCBE\Bundle\RestServerBundle\Service\Payment\Builder\BssRequestBuilder;
use WSCBE\Service\Payment\CreateOnlinePayment\PaymentDataGetter;

class PayByCardService extends AbstractPaymentService
{

    public function __construct(
        BssRequestBuilder $bssRequestBuilder,
        Resolver\RechargeResponseResolver $responseResolver,
        PaymentDataGetter $bssRechargerService
    ) {
        $this->bssRequestBuilder = $bssRequestBuilder;
        $this->responseResolver = $responseResolver;
        $this->bssPaymentDataGetter = $bssRechargerService;
    }

    public function payByCard(PayByCardRequest $request)
    {
        $onlinePayment = $this->bssRequestBuilder->buildPayByCard
($request);
        $paymentDataResult = $this->bssPaymentDataGetter-
>getOnlinePaymentData($onlinePayment);

        return $this->responseResolver->resolveFromPaymentData
($paymentDataResult);
    }
}
```

Validator

When are you validating anything. For example server request, you shall use Validator.

Validator example

```
<?php declare(strict_types=1);

namespace WSCBE\Service\PaygoTariffMigration\GetMigrationInfo\Validator;

use InvalidArgumentException;
use Vodafone\WscshopEntity\Soap\Paygo\Request\GetMigrationInfoRequest;
use Vodafone\Utility\Entity\Msisdn;

final class GetMigrationInfoRequestValidator
{
    public function validate(GetMigrationInfoRequest $request): void
    {
        $msisdn = new Msisdn($request->getServiceNumber());
        if (!$msisdn->isValid()) {
            throw new InvalidArgumentException('Invalid service
number');
        }
    }
}
```