Design for Testability

Dave Liddament
(Director and developer at Lamp Bristol Limited)

Why Test?

- Know our code works
- Prevent against regression when developing new code
- Stable platform for refactoring code

Why Design For Testability?

- Makes it easier to write tests. So more chance of testing happening.
- Easier to automate tests.
- Code that is easy to test is often better designed code.

I'm going to use HiddenCity as basis for this talk.

Treasure hunts via text message.

Play a hunt on your own or you play against other teams.

Hunt Engine is the brains of the system.

Users sign up via website. Separate front end code. Talks to Hunt Engine via web service.

Hunt Engine sends confirmation email to players.

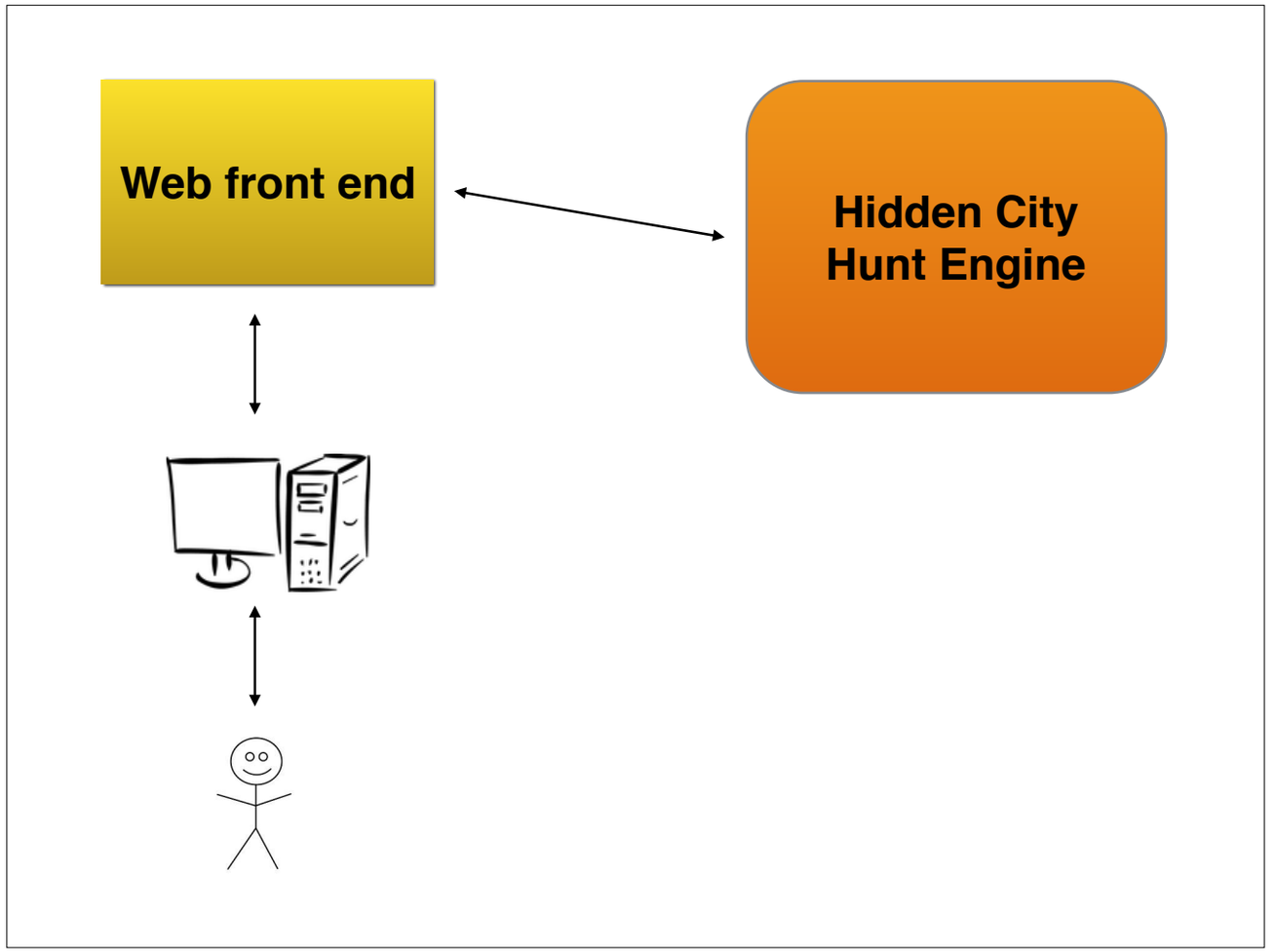
Player plays the hunt via text messages.

Hunt Engine is the brains of the system.

Users sign up via website. Separate front end code. Talks to Hunt Engine via web service.

Hunt Engine sends confirmation email to players.

Player plays the hunt via text messages.

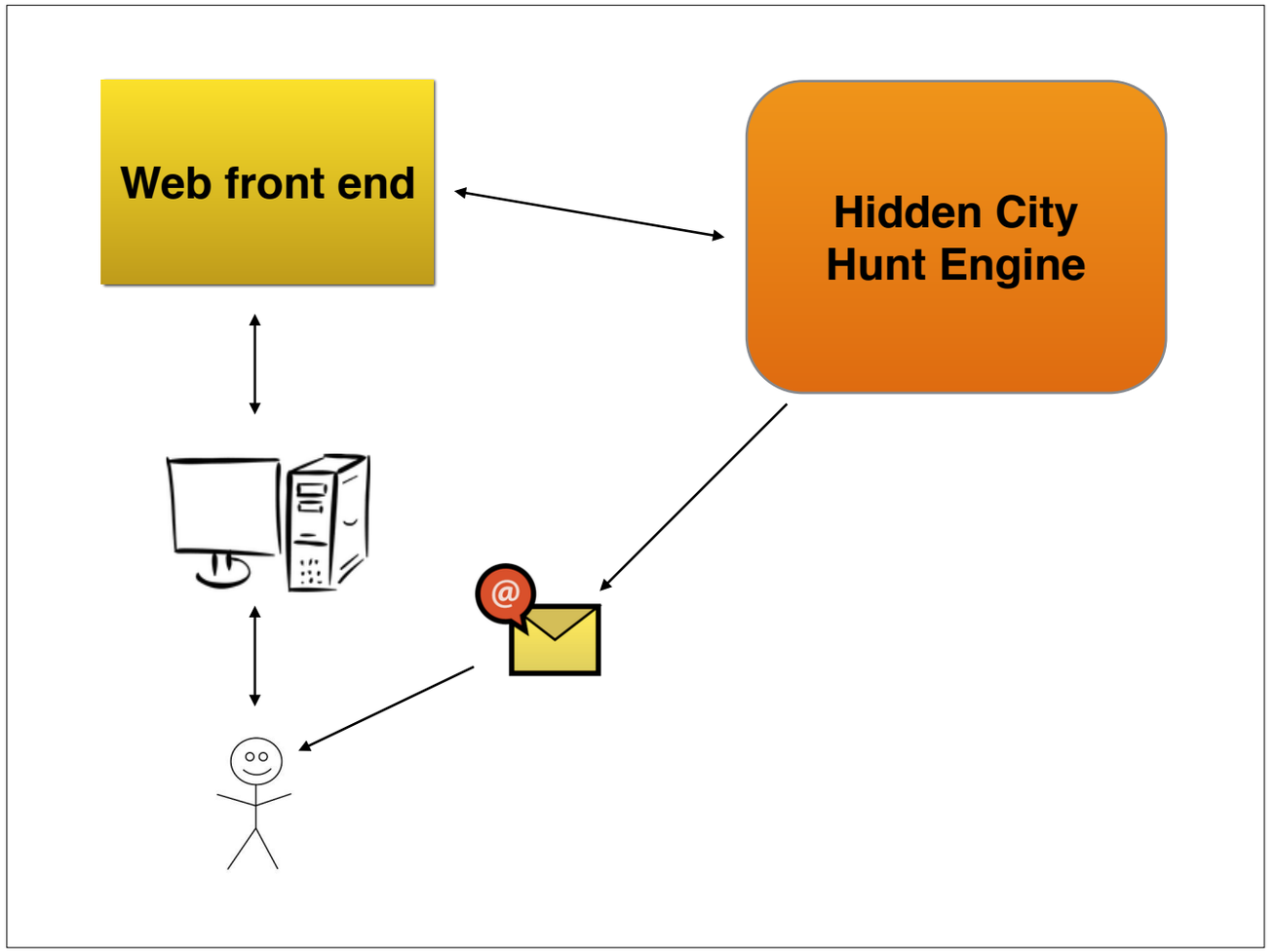


Hunt Engine is the brains of the system.

Users sign up via website. Separate front end code. Talks to Hunt Engine via web service.

Hunt Engine sends confirmation email to players.

Player plays the hunt via text messages.

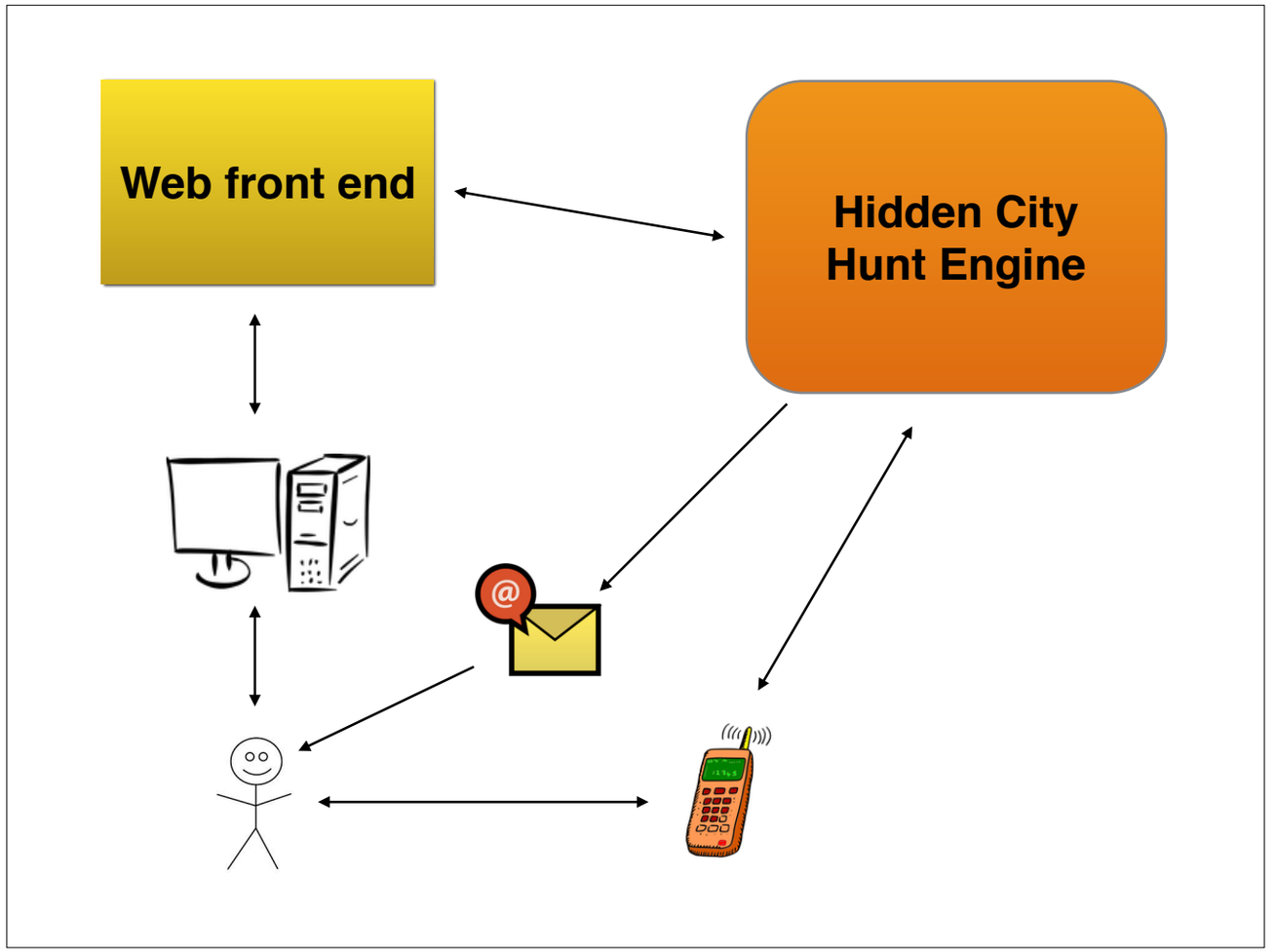


Hunt Engine is the brains of the system.

Users sign up via website. Separate front end code. Talks to Hunt Engine via web service.

Hunt Engine sends confirmation email to players.

Player plays the hunt via text messages.



Hunt Engine is the brains of the system.

Users sign up via website. Separate front end code. Talks to Hunt Engine via web service.

Hunt Engine sends confirmation email to players.

Player plays the hunt via text messages.

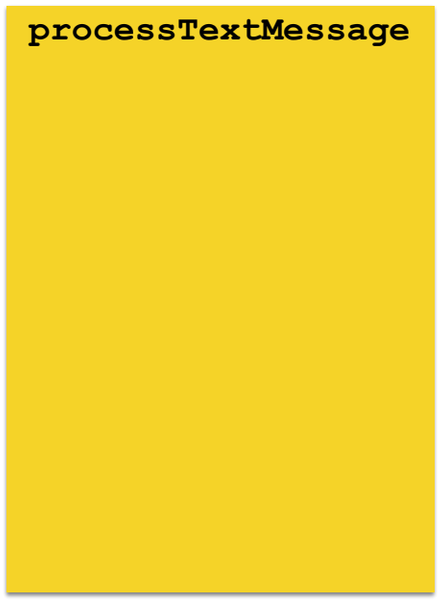
Back to the talk. Consider code to respond to a text message.

Pull out isCorrectAnswer to own function.

Split out database code to separate function.

Single Responsibility Principle & Decoupled code. Easy to unit test.

Find high value business logic and move it into easy to test function.



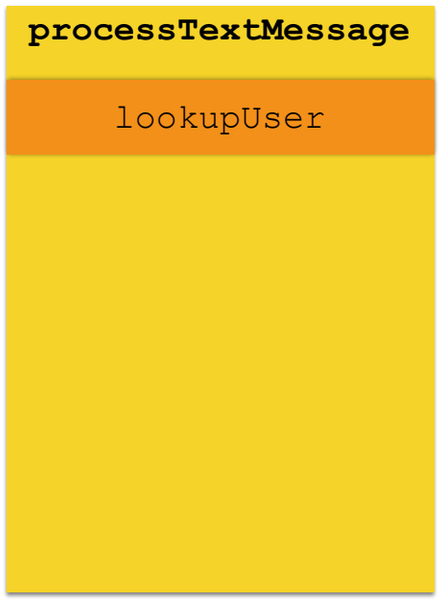
Back to the talk. Consider code to respond to a text message.

Pull out isCorrectAnswer to own function.

Split out database code to separate function.

Single Responsibility Principle & Decoupled code. Easy to unit test.

Find high value business logic and move it into easy to test function.



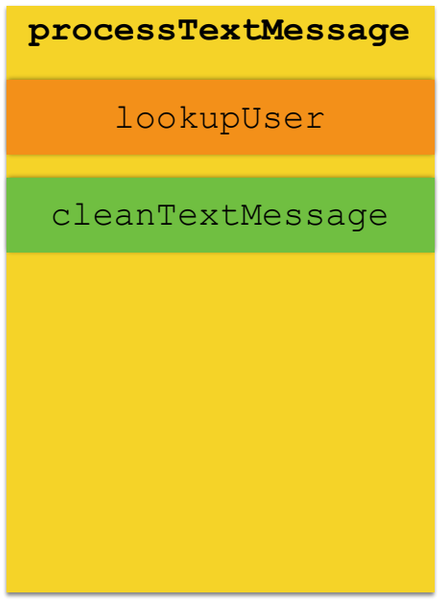
Back to the talk. Consider code to respond to a text message.

Pull out isCorrectAnswer to own function.

Split out database code to separate function.

Single Responsibility Principle & Decoupled code. Easy to unit test.

Find high value business logic and move it into easy to test function.



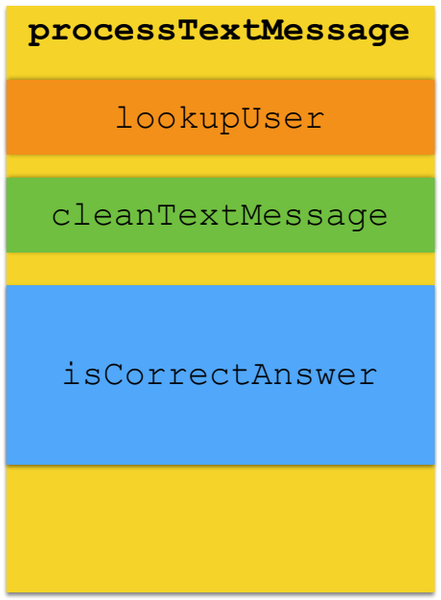
Back to the talk. Consider code to respond to a text message.

Pull out isCorrectAnswer to own function.

Split out database code to separate function.

Single Responsibility Principle & Decoupled code. Easy to unit test.

Find high value business logic and move it into easy to test function.



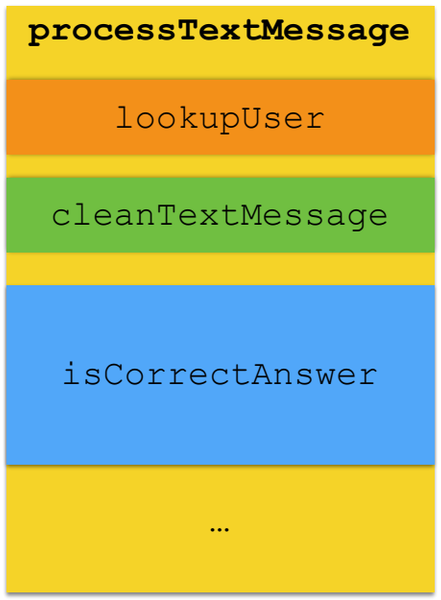
Back to the talk. Consider code to respond to a text message.

Pull out isCorrectAnswer to own function.

Split out database code to separate function.

Single Responsibility Principle & Decoupled code. Easy to unit test.

Find high value business logic and move it into easy to test function.



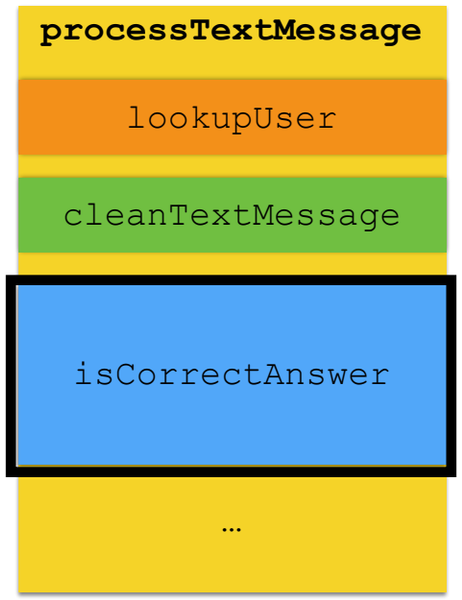
Back to the talk. Consider code to respond to a text message.

Pull out isCorrectAnswer to own function.

Split out database code to separate function.

Single Responsibility Principle & Decoupled code. Easy to unit test.

Find high value business logic and move it into easy to test function.



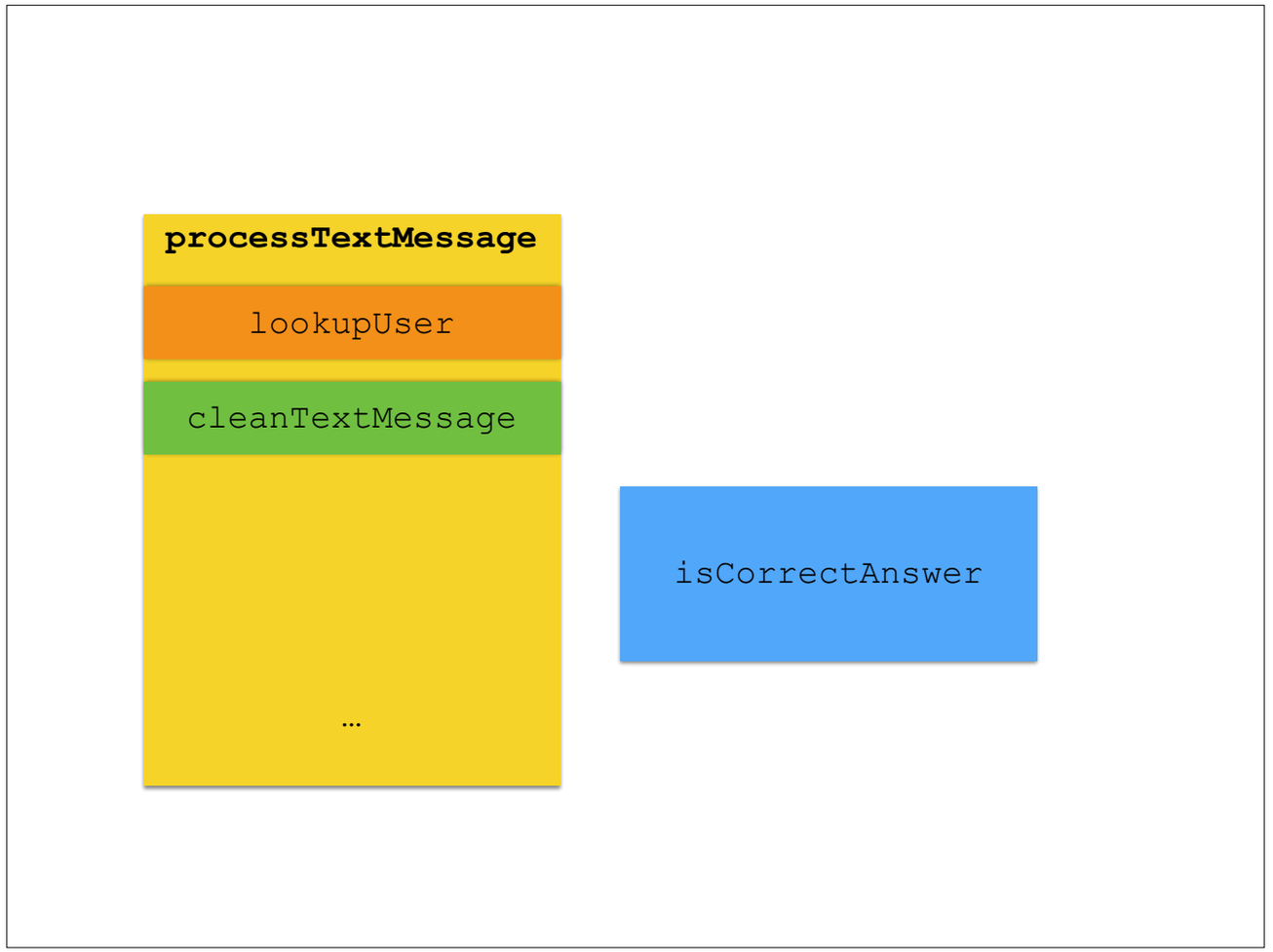
Back to the talk. Consider code to respond to a text message.

Pull out isCorrectAnswer to own function.

Split out database code to separate function.

Single Responsibility Principle & Decoupled code. Easy to unit test.

Find high value business logic and move it into easy to test function.



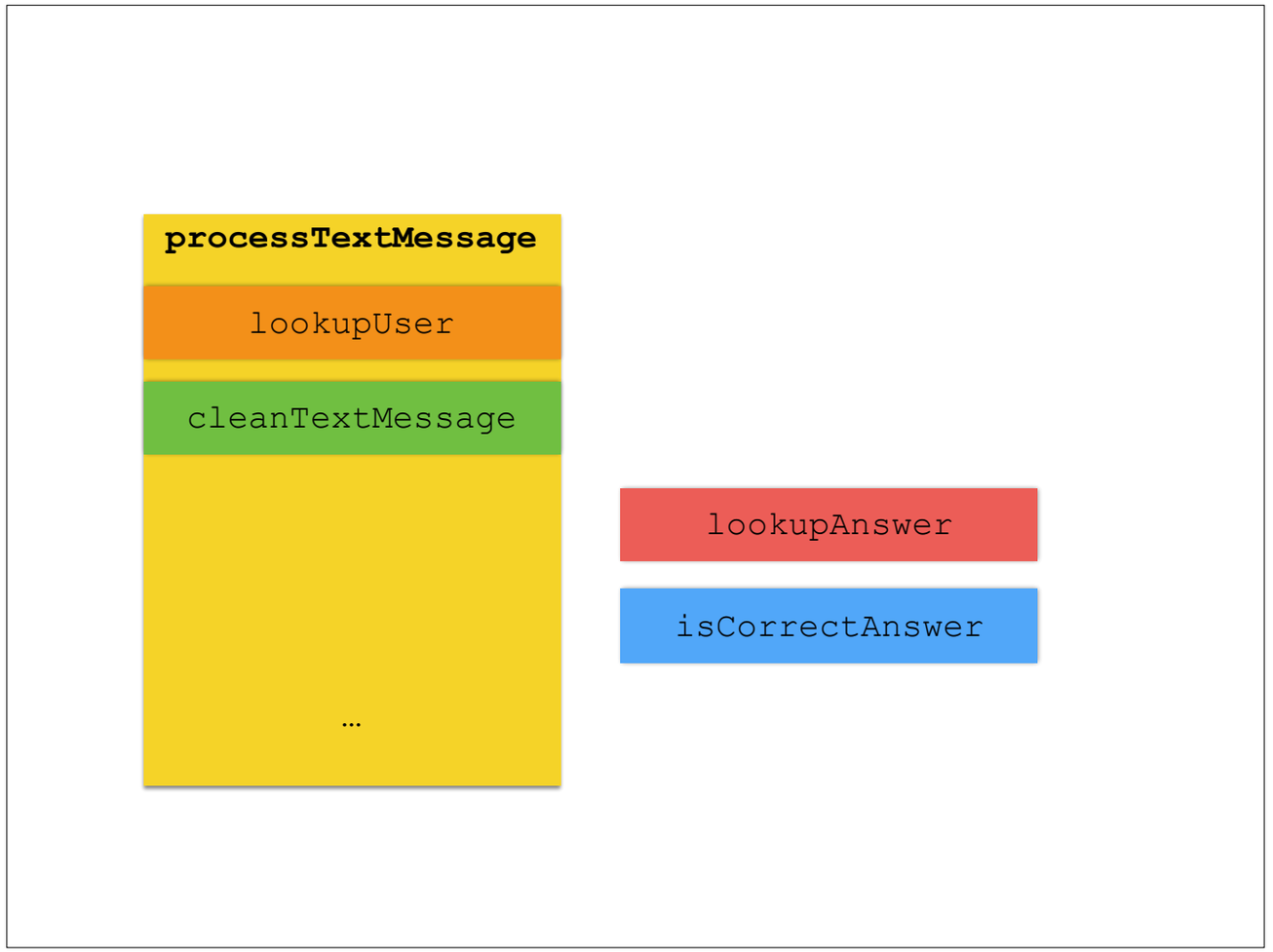
Back to the talk. Consider code to respond to a text message.

Pull out isCorrectAnswer to own function.

Split out database code to separate function.

Single Responsibility Principle & Decoupled code. Easy to unit test.

Find high value business logic and move it into easy to test function.



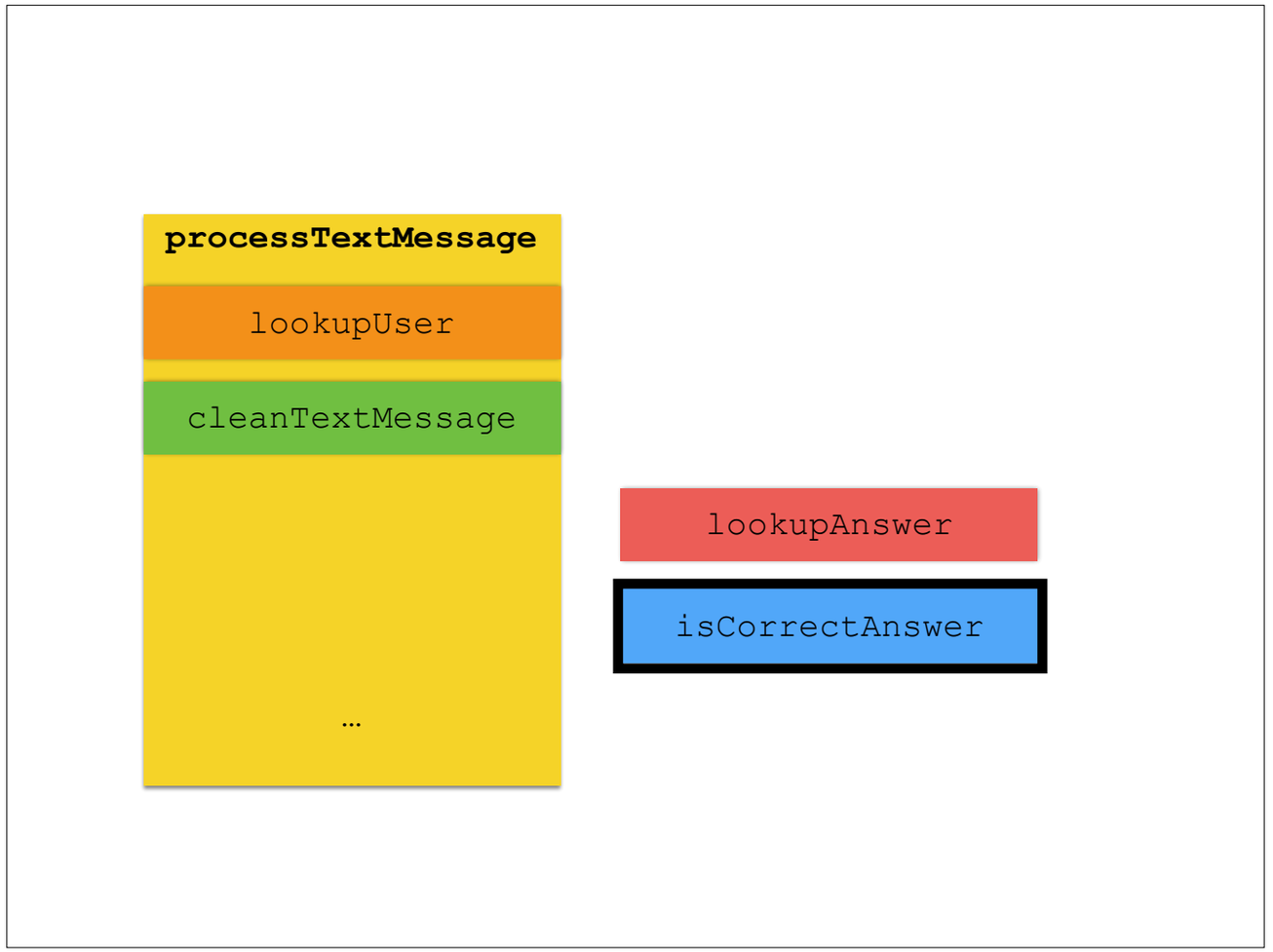
Back to the talk. Consider code to respond to a text message.

Pull out isCorrectAnswer to own function.

Split out database code to separate function.

Single Responsibility Principle & Decoupled code. Easy to unit test.

Find high value business logic and move it into easy to test function.



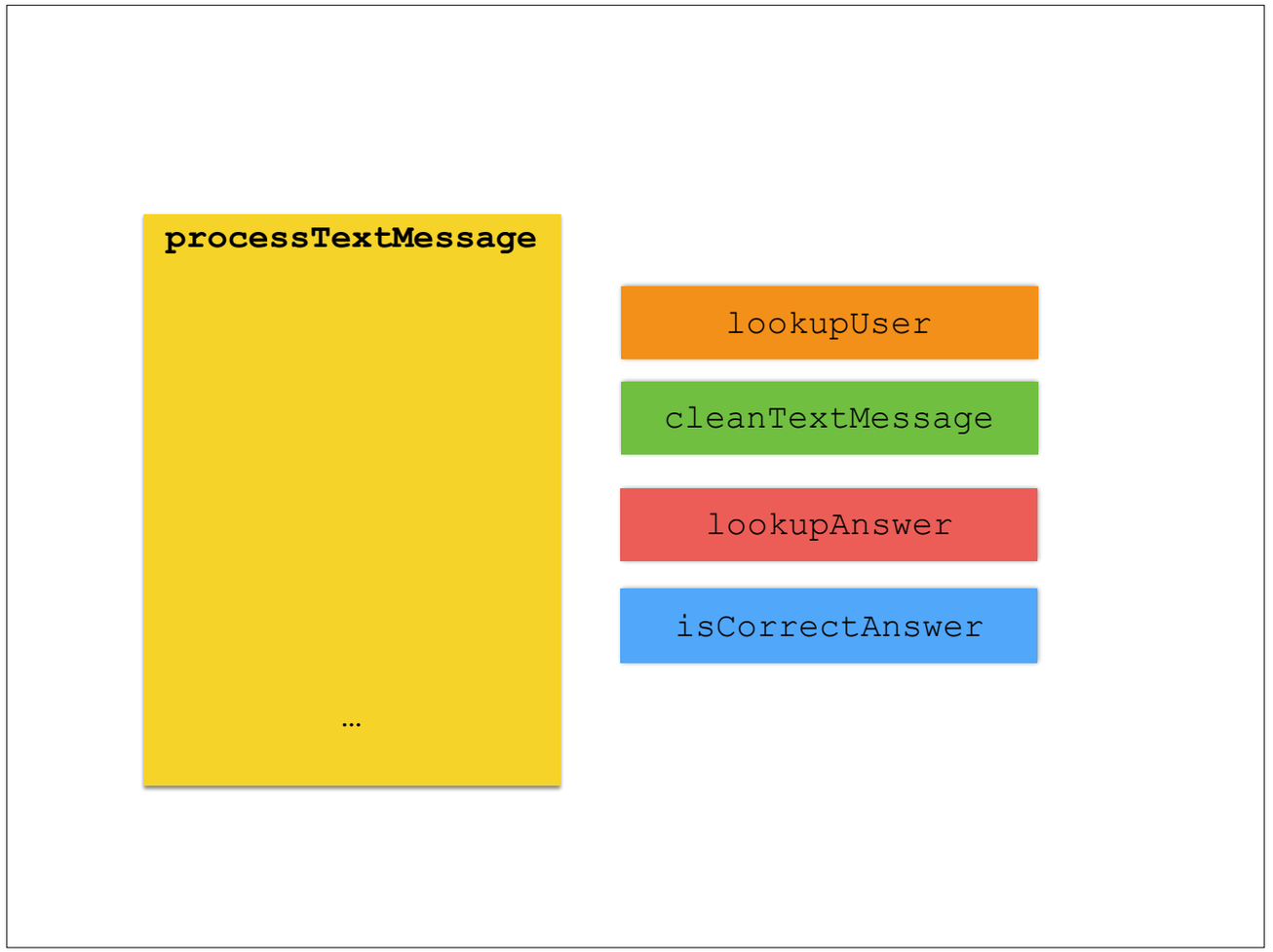
Back to the talk. Consider code to respond to a text message.

Pull out isCorrectAnswer to own function.

Split out database code to separate function.

Single Responsibility Principle & Decoupled code. Easy to unit test.

Find high value business logic and move it into easy to test function.



Back to the talk. Consider code to respond to a text message.

Pull out isCorrectAnswer to own function.

Split out database code to separate function.

Single Responsibility Principle & Decoupled code. Easy to unit test.

Find high value business logic and move it into easy to test function.

Consider this code to send text message. It’s hidden deep in Hunt Engine.

Hunt Engine fails on SRP (it should not be responsible for sending messages).

Also tightly coupled to text message code.

Very hard to test.

```
        }


        public function setTemplatePath($path) {
            $this->templatePath = $path;
        }

        private function loadMailSender() {
          if ($this->mailSender == NULL) {
                $this->mailSender = $this->configReader-
>getConfigValueAsObject("mailSender");
                $this->templateEngine = new
TemplateEnginePhpTalImpl();
                $this->mailSender->setup($this->template
                if (!isset($this->templatePath)) {
                    $this->templatePath = dirname(__FILE
"//..//templates//emails//";
                }
                $this->from = new MailContact($this-
>configReader->getConfigValue("fromName"),
                                                $this-
>configReader->getConfigValue("fromEmailAddress"));
                $this->bcc = new MailContact($this-
>configReader->getConfigValue("bccName"),
                                                $this-
```

Consider this code to send text message. It's hidden deep in Hunt Engine.

Hunt Engine fails on SRP (it should not be responsible for sending messages).

Also tightly coupled to text message code.

Very hard to test.

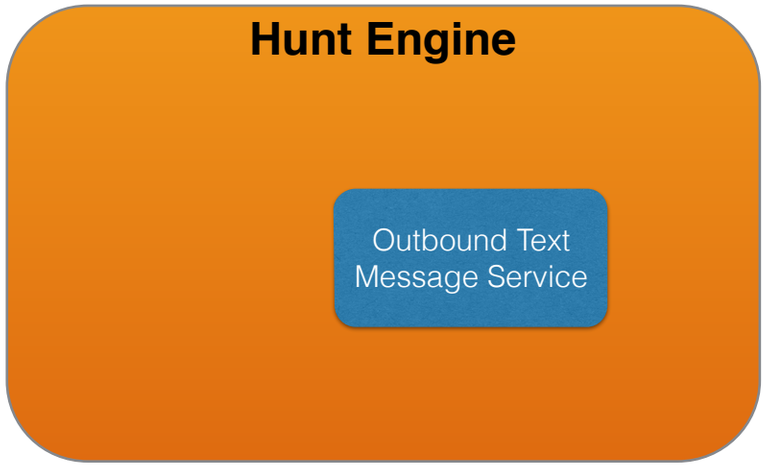
```
// Create URL to send message
$url = "http://textmessagegateway.com";
$url .= "?api_key=my_secret_key";
$url .= "&to=$number";
$url .= "&message=$message";

// Send message using cURL
$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, $url);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
$output = curl_exec($ch);
curl_close($ch);
```

Consider this code to send text message. It's hidden deep in Hunt Engine.

Hunt Engine fails on SRP (it should not be responsible for sending messages).

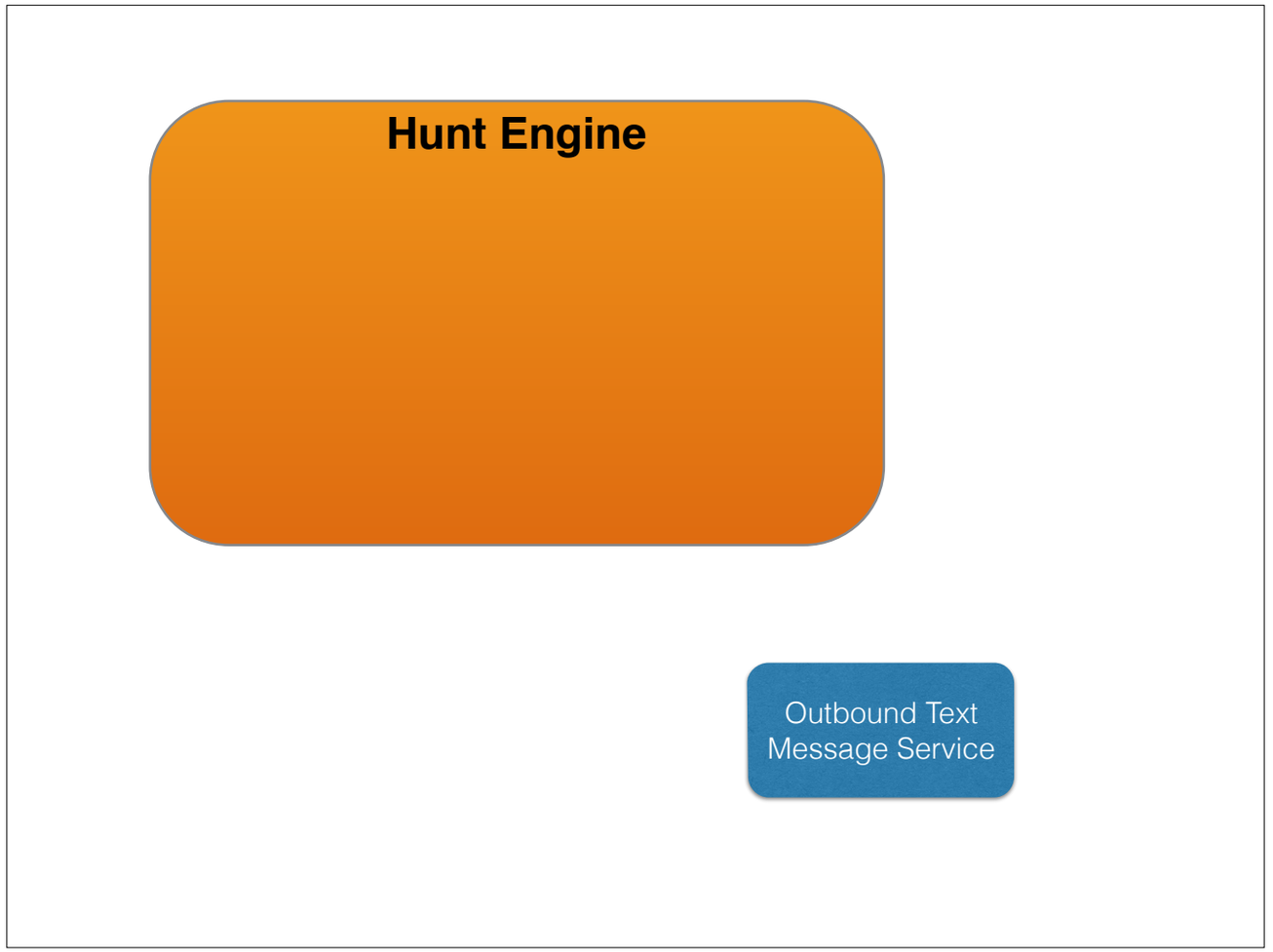
Also tightly coupled to text message code.

Very hard to test.



Need to move the code to send text message out of Hunt Engine.

Decouple text message code from Hunt Engine.



Need to move the code to send text message out of Hunt Engine.

Decouple text message code from Hunt Engine.

```php
interface OutboundTextMessageService
{

   /**
    * Send a text message
    *
    * @param $to number to send message to
    * @param $message to send
    * @throws SendingMessageException
    */
   public function sendMessage($to, $message);

}
```

All Hunt Engine cares about when sending text messages is:

- number to send message to
- message to send

```
class OutboundTextMessageServiceImpl
   implements OutboundTextMessageService
{

   public function sendMessage($to, $message)
   {
      … code from previous slide
   }

}
```

Real implementation.

Not much new code.

```
class HuntEngine
{
   private $outboundTextMessageService;

   // Constructor
   public function __construct($outbound)
   {
      $this->outboundTextMessageService = $outbound;
   }



   // Some code that uses
      $this->outboundTextMessageService
         ->sendMessage($to, 'blah blah blah');
}
```

Now hunt engine uses the OutboundTextMessageService interface to send messages.

Inject it in via constructor or set method. This is called Dependency Injection or IoC (Inversion of Control).

```
class HuntEngine
{
   private $outboundTextMessageService;

   // Constructor
   public function __construct($outbound)
   {
      $this->outboundTextMessageService = $outbound;
   }


   // Some code that uses
      $this->outboundTextMessageService
         ->sendMessage($to, 'blah blah blah');
}
```

Now hunt engine uses the OutboundTextMessageService interface to send messages.

Inject it in via constructor or set method. This is called Dependency Injection or IoC (Inversion of Control).

```
class HuntEngine
{
   private $outboundTextMessageService;

   // Constructor
   public function __construct($outbound)
   {
      $this->outboundTextMessageService = $outbound;
   }


   // Some code that uses
      $this->outboundTextMessageService
         ->sendMessage($to, 'blah blah blah');
}
```

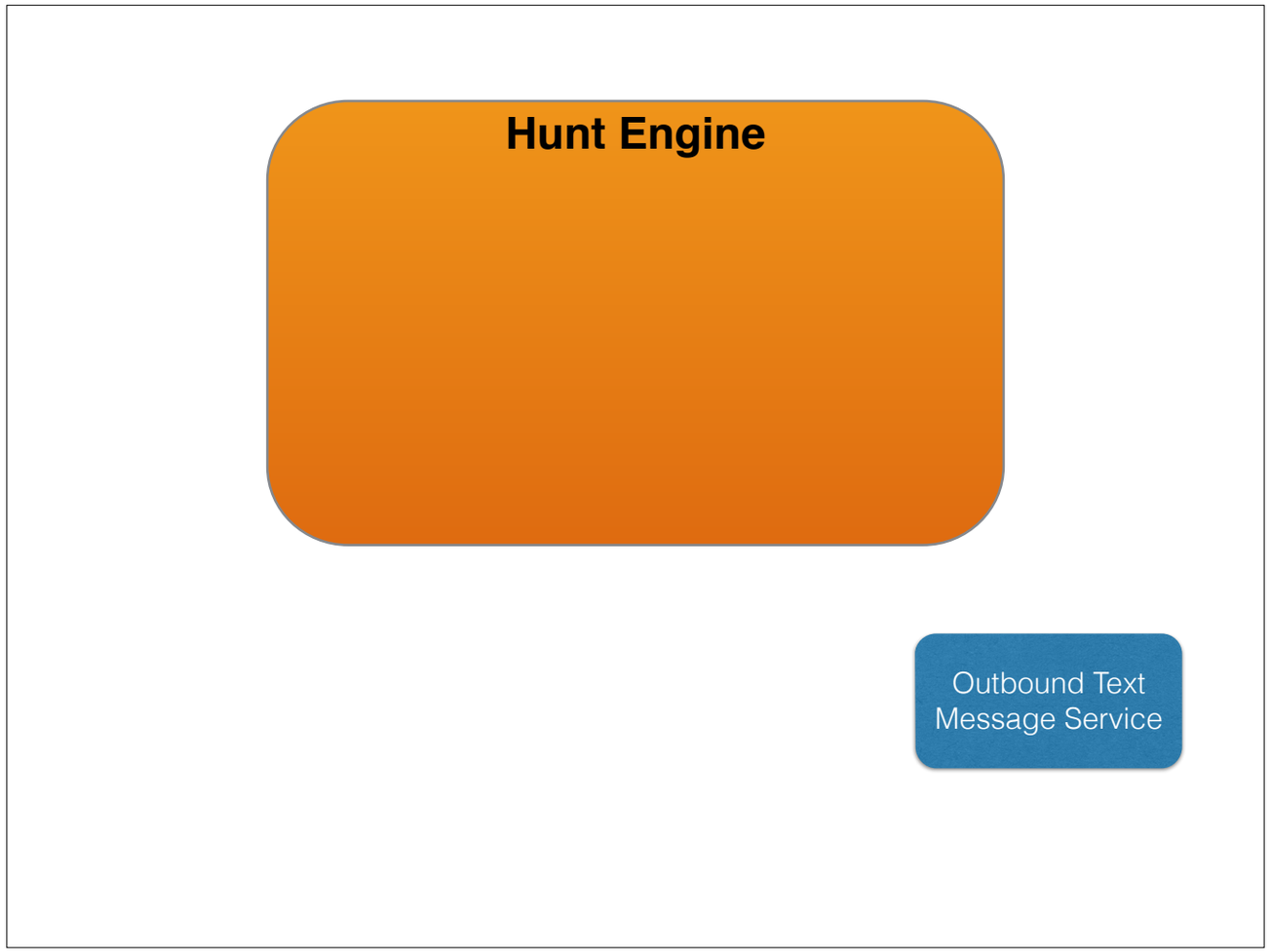
Now hunt engine uses the OutboundTextMessageService interface to send messages.

Inject it in via constructor or set method. This is called Dependency Injection or IoC (Inversion of Control).

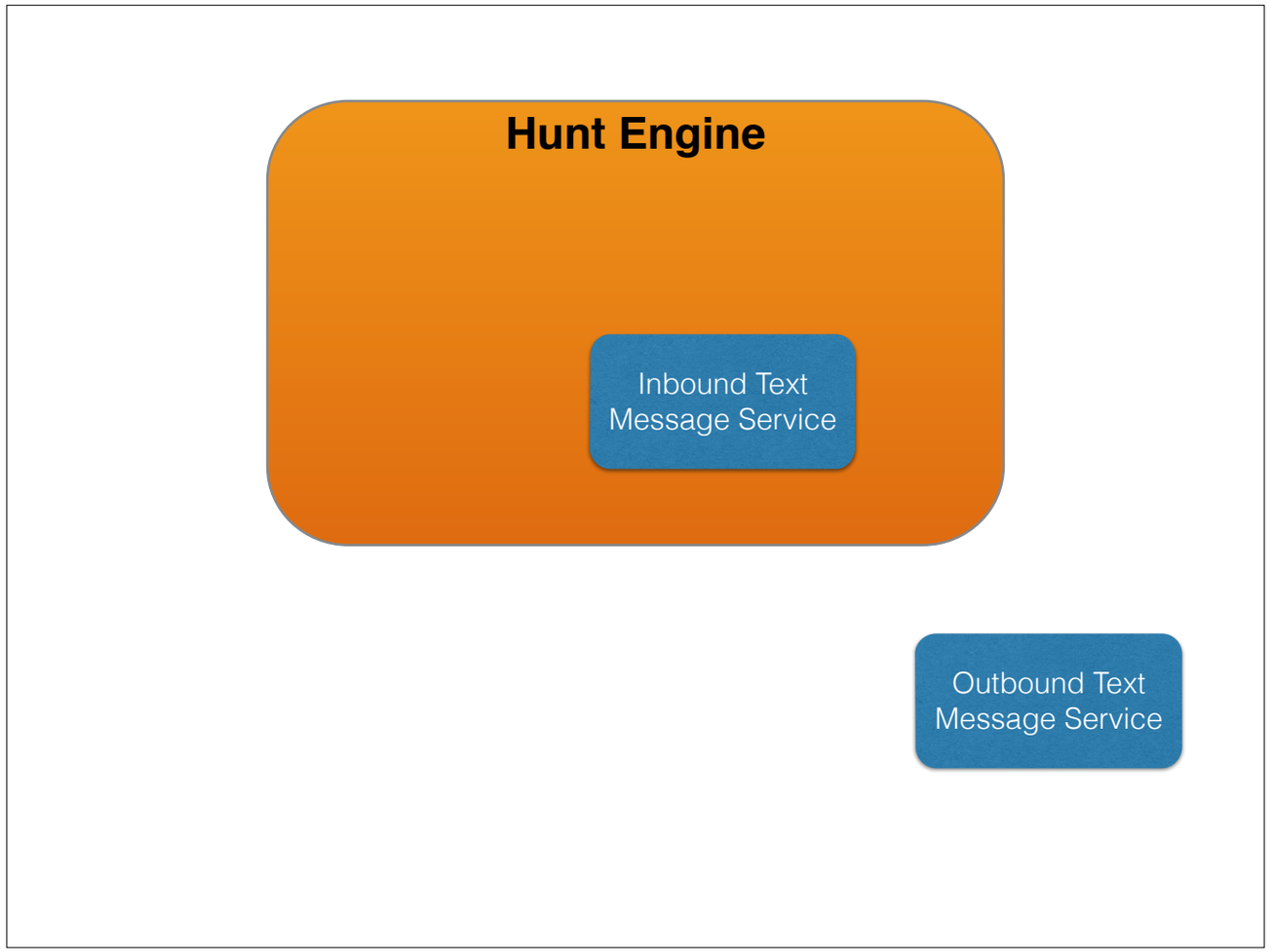
```
class HuntEngine
{
   private $outboundTextMessageService;

   // Constructor
   public function __construct($outbound)
   {
      $this->outboundTextMessageService = $outbound;
   }


   // Some code that uses
      $this->outboundTextMessageService
         ->sendMessage($to, 'blah blah blah');
}
```

Now hunt engine uses the OutboundTextMessageService interface to send messages.

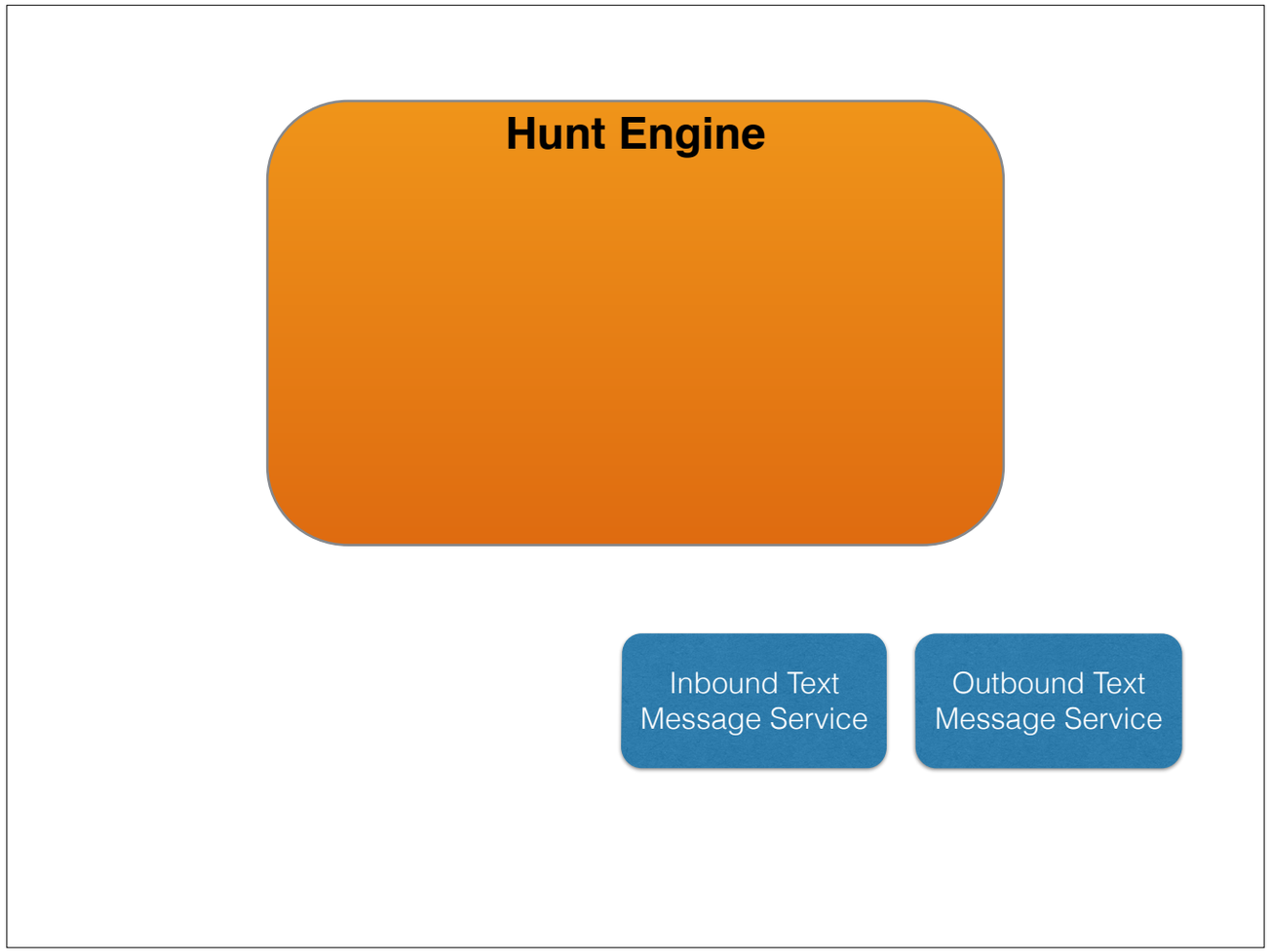
Inject it in via constructor or set method. This is called Dependency Injection or IoC (Inversion of Control).



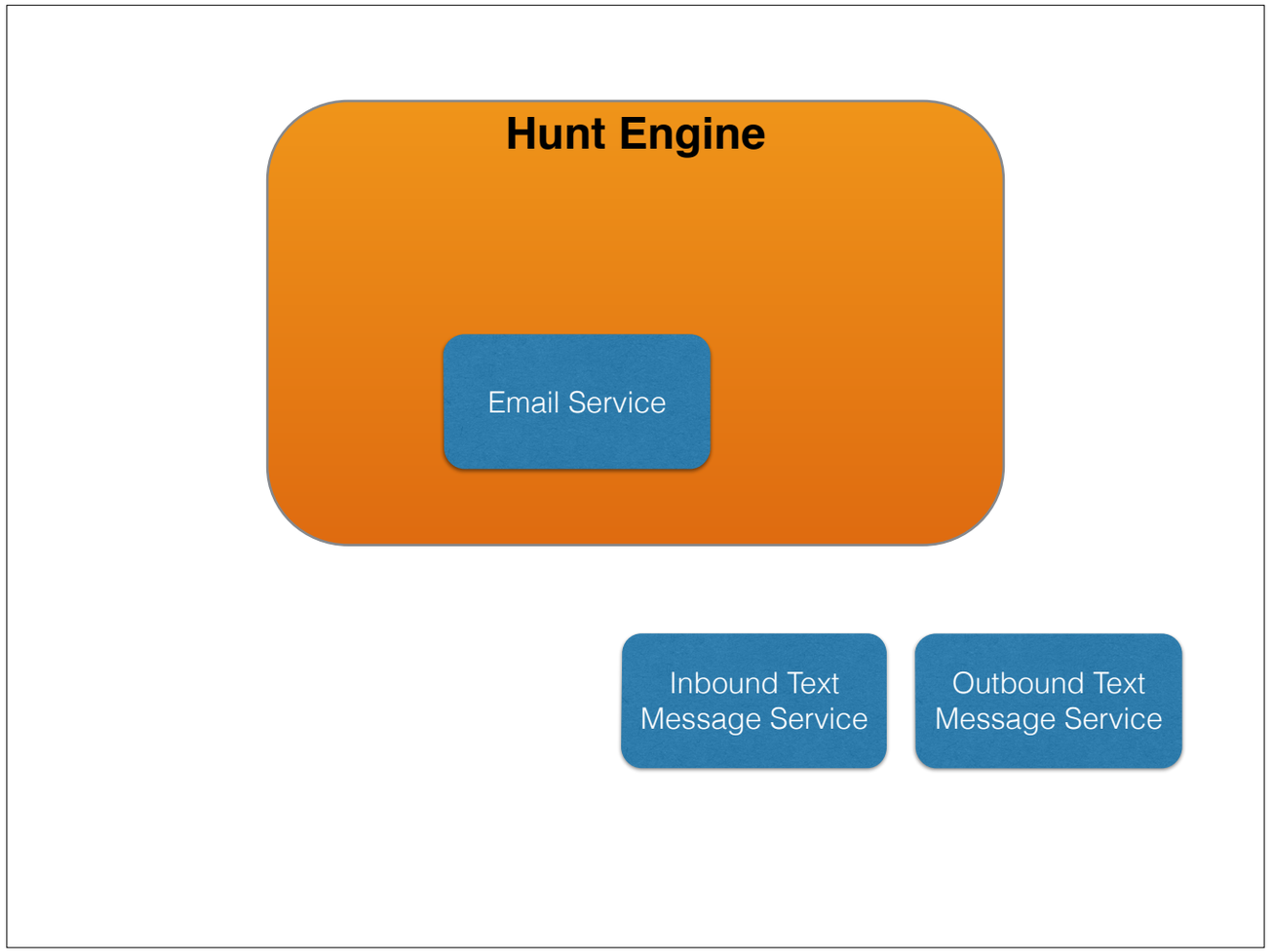
Let’s to the same with all the other external services.



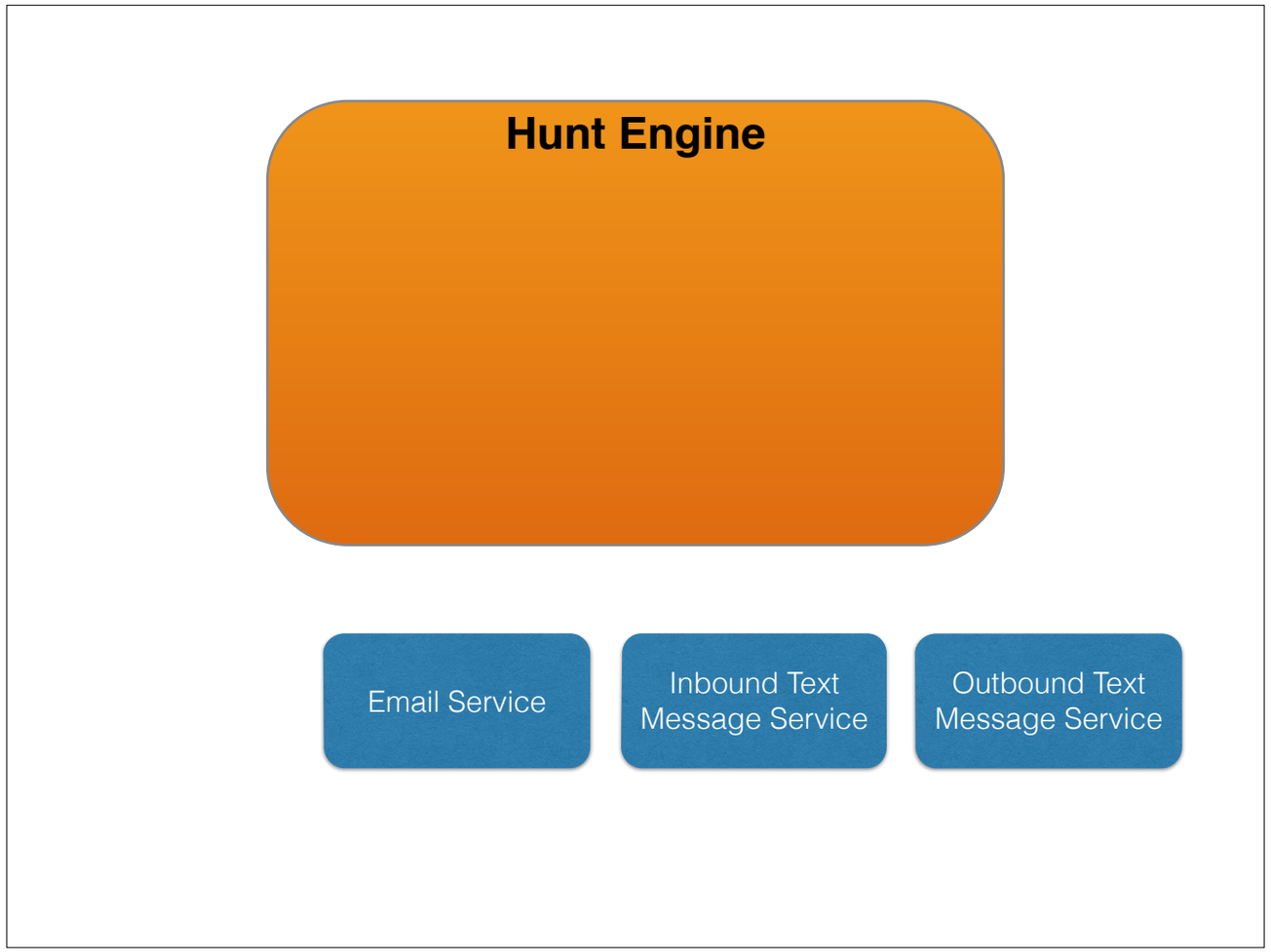
Let’s to the same with all the other external services.



Let’s to the same with all the other external services.



Let’s to the same with all the other external services.



Let’s to the same with all the other external services.

```php
interface InboundTextMessageService
{

    /**
     * Receive a text message
     *
     * @param $from number that message was from
     * @param $message
     */
    public function receiveMessage($from, $message);

}
```

Interface for inbound text messages.

All hunt engine cares about is who sent text and what the message is.

```
class DummyOutboundTextMessageServiceImpl
   implements OutboundTextMessageService
{

   public function sendMessage($to, $message)
   {
      … stores message in memory
   }

   public function getMessageCount()
   {
      … returns number of messages currently stored
   }

   public function isMessageSent($to, $message)
   {
      … returns true if message is stored,
      … also removes it from store
   }
}
```

We can now write a dummy implementation of the OutboundTextMessageService that we can use for testing.

```
class DummyOutboundTextMessageServiceImpl
   implements OutboundTextMessageService
{

   public function sendMessage($to, $message)
   {
      … stores message in memory
   }

   public function getMessageCount()
   {
      … returns number of messages currently stored
   }

   public function isMessageSent($to, $message)
   {
      … returns true if message is stored,
      … also removes it from store
   }
}
```

We can now write a dummy implementation of the OutboundTextMessageService that we can use for testing.

```
class DummyOutboundTextMessageServiceImpl
   implements OutboundTextMessageService
{

   public function sendMessage($to, $message)
   {
      … stores message in memory
   }

   public function getMessageCount()
   {
      … returns number of messages currently stored
   }

   public function isMessageSent($to, $message)
   {
      … returns true if message is stored,
      … also removes it from store
   }
}
```

We can now write a dummy implementation of the OutboundTextMessageService that we can use for testing.

```
class DummyOutboundTextMessageServiceImpl
   implements OutboundTextMessageService
{

   public function sendMessage($to, $message)
   {
      … stores message in memory
   }

   public function getMessageCount()
   {
      … returns number of messages currently stored
   }

   public function isMessageSent($to, $message)
   {
      … returns true if message is stored,
      … also removes it from store
   }
}
```

We can now write a dummy implementation of the OutboundTextMessageService that we can use for testing.

```
class HuntEngineTest extends PHPUnit_Framework_TestCase
{
   private $outbound;
   private $inbound;
   private $email;
   private $huntEngine;

   public function setUp()
   {
      $this->outbound = new DummyOutbound…();
      $this->inbound = new DummyInbound…();
      $this->email = new DummyEmail…();

      $this->huntEngine = HuntEngine(
         $this->outbound, $this->inbound,
         $this->email);
   }
```

Writing a test case…

Member data for each of the test implementations.

In setUp we inject these in to the HuntEngine.

```
class HuntEngineTest extends PHPUnit_Framework_TestCase
{
    private $outbound;
    private $inbound;
    private $email;
    private $huntEngine;

    public function setUp()
    {
        $this->outbound = new DummyOutbound…();
        $this->inbound = new DummyInbound…();
        $this->email = new DummyEmail…();

        $this->huntEngine = HuntEngine(
            $this->outbound, $this->inbound,
            $this->email);
    }
```

Writing a test case…

Member data for each of the test implementations.

In setUp we inject these in to the HuntEngine.

```
class HuntEngineTest extends PHPUnit_Framework_TestCase
{
   private $outbound;
   private $inbound;
   private $email;
   private $huntEngine;

   public function setUp()
   {
      $this->outbound = new DummyOutbound…();
      $this->inbound = new DummyInbound…();
      $this->email = new DummyEmail…();

      $this->huntEngine = HuntEngine(
         $this->outbound, $this->inbound,
         $this->email);
   }
```

Writing a test case…

Member data for each of the test implementations.

In setUp we inject these in to the HuntEngine.

```
class HuntEngineTest extends PHPUnit_Framework_TestCase
{
   private $outbound;
   private $inbound;
   private $email;
   private $huntEngine;

   public function setUp()
   {
      $this->outbound = new DummyOutbound…();
      $this->inbound = new DummyInbound…();
      $this->email = new DummyEmail…();

      $this->huntEngine = HuntEngine(
         $this->outbound, $this->inbound,
         $this->email);
   }
```

Writing a test case…

Member data for each of the test implementations.

In setUp we inject these in to the HuntEngine.

```
public function testHuntEngineWith2Teams()
{
    $this->huntEngine->signUp(ANDY, BOB, …);

    // Send 'start message'
    $this->inbound->receiveMessage(ANDY_MOBILE, 'start');

    $this->assertEquals(2, $this->outbound->getMessageCount());

    $this->assertTrue(
          $this->outbound->isMessageSent(ANDY_MOBILE, CLUE_1));

    $this->assertTrue(
          $this->outbound->isMessageSent(BOB_MOBILE, CLUE_1));

    // Bob sends in the correct answer
    $this->inbound->receiveMsg(BOB_MOBILE, 'fish');
    $this->assertEquals(1, $this->outbound->getMessageCount());
    $this->assertTrue(
          $this->outbound->isMessageSent(BOB_MOBILE, CORRECT));

    // And so on…
```

Look! All of a sudden we can actually test the Hunt Engine easily.

Andy and Bob sign up to hunt.

Start hunt, both should be sent clues.

Bob sends and answer back. Hunt Engine responds with message telling him he's got the right answer.

```
public function testHuntEngineWith2Teams()
{
    $this->huntEngine->signUp(ANDY, BOB, …);

    // Send 'start message'
    $this->inbound->receiveMessage(ANDY_MOBILE, 'start');

    $this->assertEquals(2, $this->outbound->getMessageCount());

    $this->assertTrue(
          $this->outbound->isMessageSent(ANDY_MOBILE, CLUE_1));

    $this->assertTrue(
          $this->outbound->isMessageSent(BOB_MOBILE, CLUE_1));

    // Bob sends in the correct answer
    $this->inbound->receiveMsg(BOB_MOBILE, 'fish');
    $this->assertEquals(1, $this->outbound->getMessageCount());
    $this->assertTrue(
          $this->outbound->isMessageSent(BOB_MOBILE, CORRECT));

    // And so on…
```

Look! All of a sudden we can actually test the Hunt Engine easily.

Andy and Bob sign up to hunt.

Start hunt, both should be sent clues.

Bob sends and answer back. Hunt Engine responds with message telling him he's got the right answer.

```
public function testHuntEngineWith2Teams()
{
    $this->huntEngine->signUp(ANDY, BOB, …);

    // Send 'start message'
    $this->inbound->receiveMessage(ANDY_MOBILE, 'start');

    $this->assertEquals(2, $this->outbound->getMessageCount());

    $this->assertTrue(
          $this->outbound->isMessageSent(ANDY_MOBILE, CLUE_1));

    $this->assertTrue(
          $this->outbound->isMessageSent(BOB_MOBILE, CLUE_1));

    // Bob sends in the correct answer
    $this->inbound->receiveMsg(BOB_MOBILE, 'fish');
    $this->assertEquals(1, $this->outbound->getMessageCount());
    $this->assertTrue(
          $this->outbound->isMessageSent(BOB_MOBILE, CORRECT));

    // And so on…
```

Look! All of a sudden we can actually test the Hunt Engine easily.

Andy and Bob sign up to hunt.

Start hunt, both should be sent clues.

Bob sends and answer back. Hunt Engine responds with message telling him he's got the right answer.

```
public function testHuntEngineWith2Teams()
{
    $this->huntEngine->signUp(ANDY, BOB, …);

    // Send 'start message'
    $this->inbound->receiveMessage(ANDY_MOBILE, 'start');

    $this->assertEquals(2, $this->outbound->getMessageCount());

    $this->assertTrue(
          $this->outbound->isMessageSent(ANDY_MOBILE, CLUE_1));

    $this->assertTrue(
          $this->outbound->isMessageSent(BOB_MOBILE, CLUE_1));

    // Bob sends in the correct answer
    $this->inbound->receiveMsg(BOB_MOBILE, 'fish');
    $this->assertEquals(1, $this->outbound->getMessageCount());
    $this->assertTrue(
          $this->outbound->isMessageSent(BOB_MOBILE, CORRECT));

    // And so on…
```

Look! All of a sudden we can actually test the Hunt Engine easily.

Andy and Bob sign up to hunt.

Start hunt, both should be sent clues.

Bob sends and answer back. Hunt Engine responds with message telling him he's got the right answer.

```
public function testHuntEngineWith2Teams()
{
    $this->huntEngine->signUp(ANDY, BOB, …);

    // Send 'start message'
    $this->inbound->receiveMessage(ANDY_MOBILE, 'start');

    $this->assertEquals(2, $this->outbound->getMessageCount());

    $this->assertTrue(
          $this->outbound->isMessageSent(ANDY_MOBILE, CLUE_1));

    $this->assertTrue(
          $this->outbound->isMessageSent(BOB_MOBILE, CLUE_1));

    // Bob sends in the correct answer
    $this->inbound->receiveMsg(BOB_MOBILE, 'fish');
    $this->assertEquals(1, $this->outbound->getMessageCount());
    $this->assertTrue(
          $this->outbound->isMessageSent(BOB_MOBILE, CORRECT));

    // And so on…
```

Look! All of a sudden we can actually test the Hunt Engine easily.

Andy and Bob sign up to hunt.

Start hunt, both should be sent clues.

Bob sends and answer back. Hunt Engine responds with message telling him he's got the right answer.

```
public function testHuntEngineWith2Teams()
{
    $this->huntEngine->signUp(ANDY, BOB, …);

    // Send 'start message'
    $this->inbound->receiveMessage(ANDY_MOBILE, 'start');

    $this->assertEquals(2, $this->outbound->getMessageCount());

    $this->assertTrue(
          $this->outbound->isMessageSent(ANDY_MOBILE, CLUE_1));

    $this->assertTrue(
          $this->outbound->isMessageSent(BOB_MOBILE, CLUE_1));

    // Bob sends in the correct answer
    $this->inbound->receiveMsg(BOB_MOBILE, 'fish');
    $this->assertEquals(1, $this->outbound->getMessageCount());
    $this->assertTrue(
          $this->outbound->isMessageSent(BOB_MOBILE, CORRECT));

    // And so on…
```

Look! All of a sudden we can actually test the Hunt Engine easily.

Andy and Bob sign up to hunt.

Start hunt, both should be sent clues.

Bob sends and answer back. Hunt Engine responds with message telling him he's got the right answer.

```
public function testHuntEngineWith2Teams()
{
    $this->huntEngine->signUp(ANDY, BOB, …);

    // Send 'start message'
    $this->inbound->receiveMessage(ANDY_MOBILE, 'start');

    $this->assertEquals(2, $this->outbound->getMessageCount());

    $this->assertTrue(
          $this->outbound->isMessageSent(ANDY_MOBILE, CLUE_1));

    $this->assertTrue(
          $this->outbound->isMessageSent(BOB_MOBILE, CLUE_1));

    // Bob sends in the correct answer
    $this->inbound->receiveMsg(BOB_MOBILE, 'fish');
    $this->assertEquals(1, $this->outbound->getMessageCount());
    $this->assertTrue(
          $this->outbound->isMessageSent(BOB_MOBILE, CORRECT));

    // And so on…
```

Look! All of a sudden we can actually test the Hunt Engine easily.

Andy and Bob sign up to hunt.

Start hunt, both should be sent clues.

Bob sends and answer back. Hunt Engine responds with message telling him he's got the right answer.

Now each component follows SRP.

All components are de-coupled.

Some frameworks support DI out of the box, e.g. Laravel.

```
class HuntEngineFactory
{

   public static function newHuntEngine()
   {
      return new HuntEngine(
         self::newOutboundTextMessageService(), …
      );
   }


   public static function newOutboundTextMessageService()
   {
      return new OutboundTextMessageServiceImpl();
   }
}


$huntEngine = HuntEngineFactory::newHuntEngine();
```

How do we create a HuntEngine object if not using DI framework?

Could use simple factory.

Think SRP - newHuntEngine() only builds hunt engine, it delegates to other methods for components it needs.

Not only is code easy to test, just consider how easy it would be able to swap to different outbound message services.

```
class HuntEngineFactory
{
   public static function newHuntEngine()
   {
      return new HuntEngine(
         self::newOutboundTextMessageService(), …
      );
   }

   public static function newOutboundTextMessageService()
   {
      return new OutboundTextMessageServiceImpl();
   }
}


$huntEngine = HuntEngineFactory::newHuntEngine();
```

How do we create a HuntEngine object if not using DI framework?

Could use simple factory.

Think SRP - newHuntEngine() only builds hunt engine, it delegates to other methods for components it needs.

Not only is code easy to test, just consider how easy it would be able to swap to different outbound message services.

```
class HuntEngineFactory
{

   public static function newHuntEngine()
   {
      return new HuntEngine(
         self::newOutboundTextMessageService(), …
      );
   }

   public static function newOutboundTextMessageService()
   {
      return new OutboundTextMessageServiceImpl();
   }
}


$huntEngine = HuntEngineFactory::newHuntEngine();
```

How do we create a HuntEngine object if not using DI framework?

Could use simple factory.

Think SRP - newHuntEngine() only builds hunt engine, it delegates to other methods for components it needs.

Not only is code easy to test, just consider how easy it would be able to swap to different outbound message services.

```
class HuntEngineFactory
{

   public static function newHuntEngine()
   {
      return new HuntEngine(
         self::newOutboundTextMessageService(), …
      );
   }


   public static function newOutboundTextMessageService()
   {
      return new OutboundTextMessageServiceImpl();
   }
}
```

```
$huntEngine = HuntEngineFactory::newHuntEngine();
```

How do we create a HuntEngine object if not using DI framework?

Could use simple factory.

Think SRP - newHuntEngine() only builds hunt engine, it delegates to other methods for components it needs.

Not only is code easy to test, just consider how easy it would be able to swap to different outbound message services.

Summary

- Think about testing from the start
- Single Responsibility Principle
- Loose coupling
- Defined interfaces
- Dependency Injection / IoC