Sperasoft

Your game dev partner

Mocks Introduction

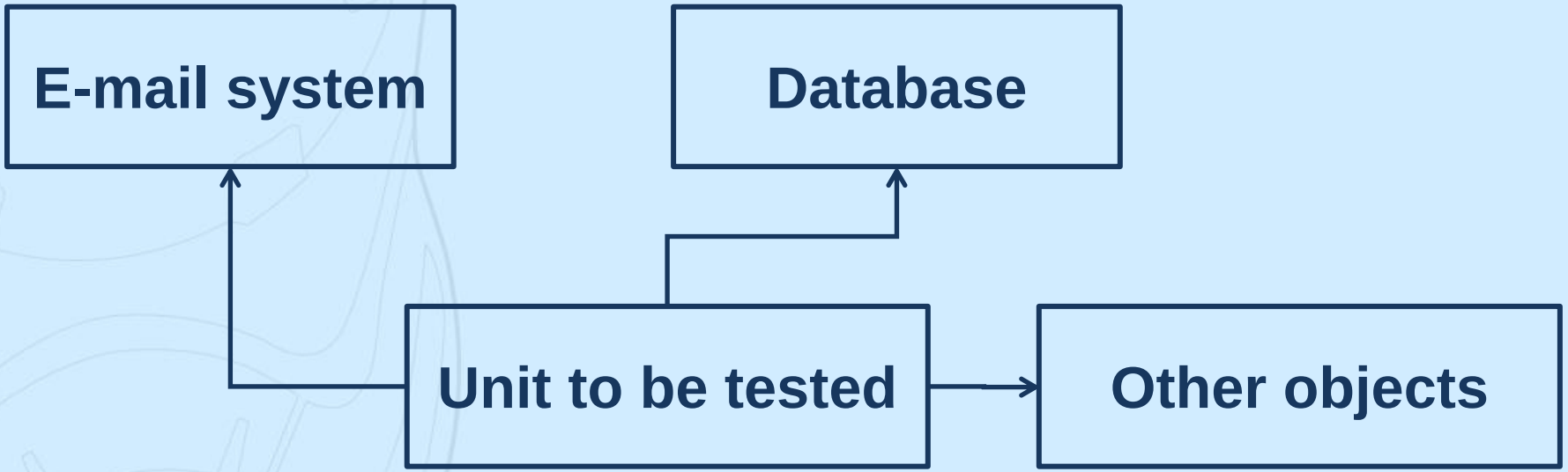
Contents

Sperasoft
Your game dev partner

- Classic style
- Mockist style
- Different tools
- Links

Classic Style

- State verification
- Heavyweight dependencies
- Test doubles (Dummy, Fake, Stub, Mock)



Example. Dog and House

Dog Class

```
package com.sperasoft.test;

public class Dog {

	private House home;

	public boolean isPleasured() {

		if (home == null) {
			return false;
		}

		return (home.getWidth() * home.getHeight() *
home.getDepth() > 3);
	}

	public void settleIn(House home) {
		this.home = home;
	}
}
```

House Interface

```
package com.sperasoft.test;

public interface House {

		int getWidth();

		int getHeight();

		int getDepth();

}
```

House Implementation

Sperasoft
Your game dev partner

```
package com.sperasoft.test;

public class HouseImpl implements House {

        private int w;
        private int h;
        private int d;

        public HouseImpl(int width, int height, int depth) {
                w = width;
                h = height;
                d = depth;
        }

        public int getWidth() {
                return w;
        }

        public int getHeight() {
                return h;
        }

        public int getDepth() {
                return d;
        }

}
```

Classic Test

```
package com.sperasoft.test;

import static org.junit.Assert.*;

import org.junit.Test;

public class DogTest {

        @Test
        public void testIsPleasuredWithBigHouse() {

                Dog dog = new Dog();

                House dogHouse = new HouseImpl(1, 2, 3);

                dog.settleIn(dogHouse);

                assertTrue(dog.isPleasured());

        }

        //other test methods
}
```

Classic Test with Stub

```
package com.sperasoft.test;

import static org.junit.Assert.*;

import org.junit.Test;

public class DogTest {

        @Test
        public void testIsPleasuredWithBigHouse() {

                Dog dog = new Dog();

                House dogHouse = new House() {

                        public int getWidth() {
                                return 1;
                        }

                        public int getHeight() {
                                return 2;
                        }

                        public int getDepth() {
                                return 3;
                        }
                };

                dog.settleIn(dogHouse);

                assertTrue(dog.isPleasured());
        }

        //other test methods
}
```

Test with Mocks

```
package com.sperasoft.test;

import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;

import mockit.Mocked;
import mockit.NonStrictExpectations;

import org.junit.Test;

public class DogTestJMockit {

	@Mocked
	private House houseMock;

	@Test
	public void testIsPleasuredWithBigHouse() {

		new NonStrictExpectations() {
			{
				houseMock.getWidth(); result
= 1;
				houseMock.getHeight(); result
= 2;
				houseMock.getDepth(); result
= 3; maxTimes = 1;
			}
		};

		Dog dog = new Dog();

		dog.settleIn(houseMock);

		assertTrue(dog.isPleasured());
	}

	//other test methods
}
```

Weather!

```
package com.sperasoft.test;

final public class Weather {

	static public int getTemperature() {

		return (int)(Math.random()*60 -
20);

	}

}
```

Dog & Weather

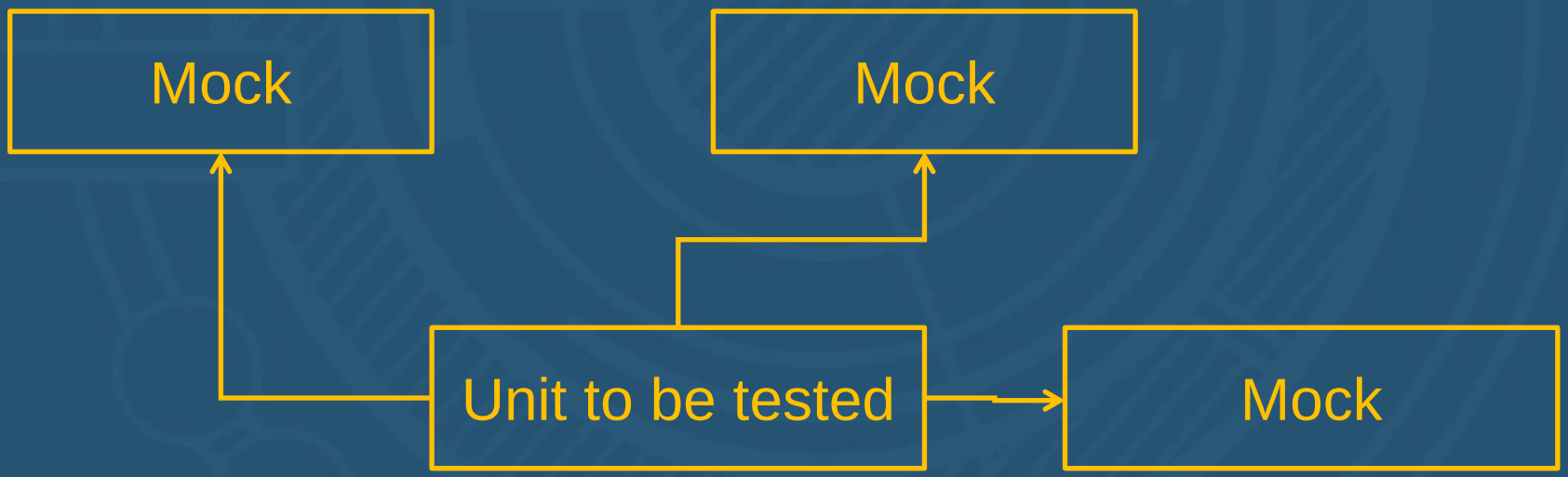
```
package com.sperasoft.test;

public class Dog {

	private House home;

	public boolean isPleasured() {

		if (home == null) {
			return false;
		}

		if (Weather.getTemperature() > 30 || Weather.getTemperature() < 20)
{
			return false;
		}

		return (home.getWidth() * home.getHeight() * home.getDepth() > 3);
	}

	public void settleIn(House home) {
		this.home = home;
	}
}
```

Mocking Weather

```
package com.sperasoft.test;

//...imports

import mockit.Mocked;
import mockit.NonStrictExpectations;

public class DogTestJMockit {

        @Mocked
        private House houseMock;

        @Mocked
        private Weather weatherMock;

        @Test
        public void testIsPleasuredWithBigHouse() {

                new NonStrictExpectations() {
                        {
                                houseMock.getWidth(); result = 1;
                                houseMock.getHeight(); result = 2;
                                houseMock.getDepth(); result = 3; maxTimes = 1;

                                Weather.getTemperature(); result = 25; times = 1;
                        }
                };

                Dog dog = new Dog();

                dog.settleIn(houseMock);

                assertTrue(dog.isPleasured());
        }

        //other test methods
}
```

Mocks Style

- Setup and verification are extended by expectations
- Behaviour verification
- Need Driven Development
- Test spies alternative (stubs with behaviour verifications)



Advantages of Mocks

- Immediate neighbours only
- Outside-in style
- Test isolation
- Good to test objects that don't change their state

Disadvantages of Mocks

Sperasoft
Your game dev partner

- Additional knowledge
- Difficult to maintain
- Heavy coupling to an implementation
  Have to use TDD. Tests first. Use loose expectations to avoid this.
- Overhead additional libraries settings
- Addictive

EasyMock

Sperasoft
Your game dev partner

```
package com.sperasoft.test;

import static org.junit.Assert.assertTrue;

import org.easymock.EasyMock;
import org.junit.Test;

public class DogTestEasyMock {

        @Test
        public void testIsPleasuredWithBigHouse() {
                Dog dog = new Dog();

                House dogHouse = EasyMock.createMock(House.class);

                EasyMock.expect(dogHouse.getWidth()).andReturn(1);
                EasyMock.expect(dogHouse.getHeight()).andReturn(2);
                EasyMock.expect(dogHouse.getDepth()).andReturn(3).times(0, 1);

                EasyMock.replay(dogHouse);

                dog.settleIn(dogHouse);

                assertTrue(dog.isPleasured());
        }

        //other test methods
}
```

JMock

Sperasoft
Your game dev partner

```
package com.sperasoft.test;

import static org.junit.Assert.assertTrue;

import org.jmock.Expectations;
import org.jmock.Mockery;
import org.junit.Test;

public class DogTestJMock {

        @Test
        public void testIsPleasuredWithBigHouse() {

                Mockery context = new Mockery();

                Dog d = new Dog();

                final House dogHouse = context.mock(House.class);

                Expectations expectations = new Expectations() {
                        {
                                allowing(dogHouse).getWidth();
                                will(returnValue(1));

                                allowing(dogHouse).getHeight();
                                will(returnValue(2));

                                oneOf(dogHouse).getDepth();
                                will(returnValue(3));
                        }
                };
                context.checking(expectations);

                d.settleIn(dogHouse);

                assertTrue(d.isPleasured());
        }
}
```

Mockito

```java
package com.sperasoft.test;

import static org.junit.Assert.assertTrue;

import org.junit.Test;
import org.mockito.Mockito;

public class DogTestMockito {

        @Test
        public void testIsPleasuredWithBigHouse() {
                Dog dog = new Dog();

                House dogHouse = Mockito.mock(House.class);

                Mockito.when(dogHouse.getWidth()).thenReturn(1);
                Mockito.when(dogHouse.getHeight()).thenReturn(2);
                Mockito.when(dogHouse.getDepth()).thenReturn(3);

                dog.settleIn(dogHouse);

                assertTrue(dog.isPleasured());

                Mockito.verify(dogHouse, Mockito.times(1)).getDepth();
        }

        //other test methods
}
```

Links

- **M. Fowler «Mocks aren't stubs»**
  http://martinfowler.com/articles/mocksArentStubs.html
- **Gerard Meszaros's book «Xunit test patterns»**
  http://xunitpatterns.com/
- **Dan North's Behaviour Driven Development**
  http://dannorth.net/introducing-bdd/
- **Jmock.** *http://www.jmock.org/*
- **EasyMock** *http://easymock.org/*
- **Mockito** *http://code.google.com/p/mockito/*
- **Jmockit** *http://code.google.com/p/jmockit/*

Questions?