AARHUS UNIVERSITET

# Software Engineering and Architecture

## Test Spy Revisited

## Unit Testing Strategies

AARHUS UNIVERSITET

# Problem Statement

- *Some strategies can be **Unit Tested***
  - My ‘ManaProductionStrategy’ does not depend upon any HotStone abstraction

```
/** The Beta specification. */
public class OneManaPerRoundStrategy implements ManaProductionStrategy {
  @Override  16 usages  Henrik Bærbak Christensen
  public int getManaCountForTurn(int turnCount) {
    int mana = (turnCount / 2) +1;
    return mana > 7 ? 7 : mana;
  }
}
```

```
@Test  Henrik Bærbak Christensen
public void shouldValidateOneManaRoundStrategy() {
  ManaProductionStrategy strategy = new OneManaPerRoundStrategy();
  assertThat(strategy.getManaCountForTurn( turnCount: 1), is( value: 1));
  assertThat(strategy.getManaCountForTurn( turnCount: 2), is( value: 2));
  assertThat(strategy.getManaCountForTurn( turnCount: 3), is( value: 2));
  assertThat(strategy.getManaCountForTurn( turnCount: 4), is( value: 3));
  assertThat(strategy.getManaCountForTurn( turnCount: 6), is( value: 4));

  assertThat(strategy.getManaCountForTurn( turnCount: 11), is( value: 6));
  assertThat(strategy.getManaCountForTurn( turnCount: 13), is( value: 7));
  assertThat(strategy.getManaCountForTurn( turnCount: 14), is( value: 7));
  assertThat(strategy.getManaCountForTurn( turnCount: 87), is( value: 7));
}
```

- So I can unit test it: in *isolation* from other objects…

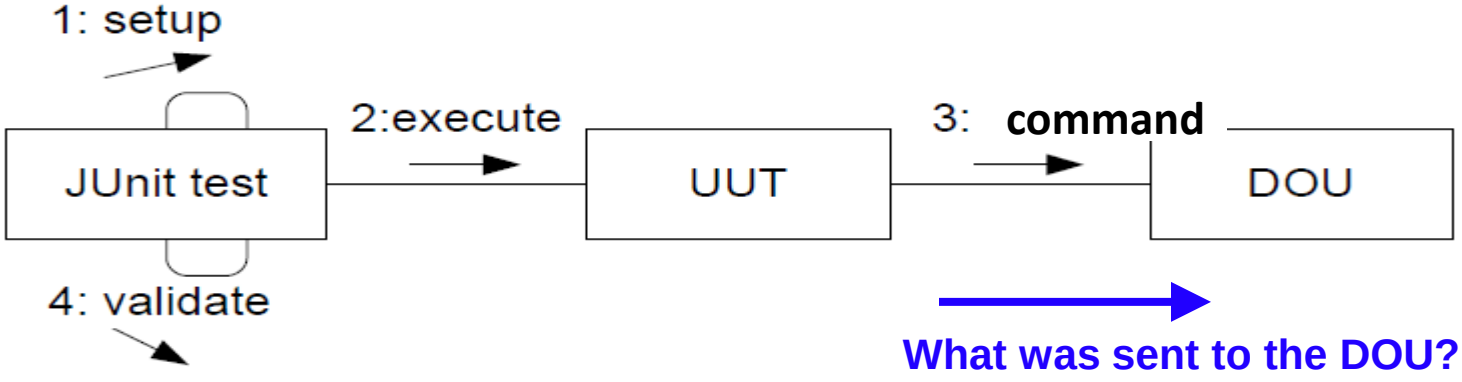
AARHUS UNIVERSITET

# Problem Statement

- GammaStone was a problem – no **Unit Testing** possible of the Hero Power Strategy
  - It modifies the game object ala reducing health of hero by two…
    - … which of course require us to have a game object in place
- Ala an **integration** test case like
  - GIVEN a GammaStone Game
  - WHEN I ask Thai Hero (Findus) to ‘use your power’
  - THEN game.getHero().getHealth() is reduced by 2
- Integration test because
  - Both Game and Hero Power Strategy (and Hero?) involved…

# Analysis

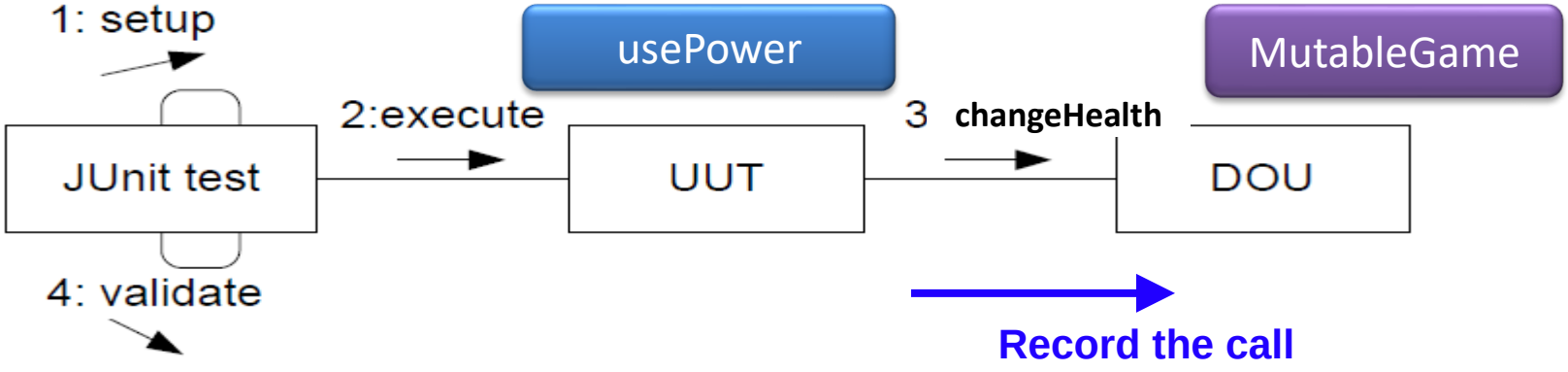
- Seen from the perspective of the hero power strategy
  - *The usePower() method mutates a Depended Upon Unit (DOU)*
- Spies serve ***commands (mutators)*** by the UUT




- Spies are *recorders* of interaction
  - So JUnit test can *later query the spy about "what happened?"*

# Analysis

- A Double/Spy is a *replacement* of the original DOU
  - Which requires that the DOU is defined by an interface/role
- **Now our private interface for Game allows us to do unit testing!** Replace MutableGame by a spy



AARHUS UNIVERSITET

# My Code

- A Unit testing of Thai Chef, by using a Spy

```
@Test  Henrik Bærbak @ coffeelake.small22 <hbc@cs.au.dk>
public void shouldDealTwoDamageToOpponentHeroWhenThaiChefUsePower() {
  // Given a SPY on MutableGame
  SpyMutableGame spy = new SpyMutableGame();
  // Given a Thai+Danish chef Hero building strategy
  HeroBuildingStrategy heroBuildingStrategy = new ThaiDanishChefHeroBuildingStrategy();
  // Given a Thai chef (Findus plays Thai)
  Hero thaiHero = heroBuildingStrategy.createHero(Player.FINDUS);

  // When Thai chef executes its power
  thaiHero.getEffect().executeEffect(spy, 0 /* no care */);

  // Then our MutableGame is told to reduce health of Peddersen by two
  assertThat(spy.lastCall, is( value: "deltaHeroHealth(PEDDERSEN,-2)"));
}
```

# My Spy

- Is a simple ‘record last method call’

```
class SpyMutableGame implements InternalMutableGame {

  public String lastCall = "none";  5 usages
```

```
@Override   Henrik Bærbak @ coffeelake.small22 <hbc@cs.au.dk>
public void deltaHeroHealth(Player who, int value) {
  lastCall = "deltaHeroHealth(" + who + "," + value + ")";
}
```

AARHUS UNIVERSITET

# Discussion

- Note how all these techniques combine to make it possible
  - Interface Segregation Principle + Role/Private interface
    - Game object can play both an “outward looking” and “inward looking” role
      - Game interface: outward looking what outsiders can do
      - MutableGame: inward looing what strategies can do
  - *Program to an interface*
    - MutableGame is an interface, and can be *played by another object than the real implementation itself*
  - Test Spy
    - The spy plays the MutableGame role, to test the hero power algorithm

AARHUS UNIVERSITET

# Discussion

- Unit Testing changes the GWT ‘mind set’
- From
  - Given **game**; when execute power; then assert **state of game**
- To
  - Given **strategy;** when execute power; then assert **proper mutator method of ‘mutable game’ role is called with the proper parameters**

```
    // Then our MutableGame is told to reduce health of Peddersen by two
    assertThat(spy.lastCall, is( value: "deltaHeroHealth(PEDDERSEN,-2)"));
}
```

AARHUS UNIVERSITET

# Benefit/Liability

- Liabilities
  - You have to code the Spy
    - (Mock frameworks like ‘Mockito’ may reduce that effort)
- Benefits
  - Much more evident tests
  - Much shorter tests

```
@Test  Henrik Bærbak @ coffeelake.small22 <hbc@cs.au.dk>
public void shouldExecuteBrownRiceEffect() {
  // Given the EtaStone BrownRice card
  Card card = theDeck.get(0);
  assertThat(card.getName(), is(GameConstants.BROWN_RICE_CARD));
  // When executing the card effect
  card.getEffect().executeEffect(spy, dropIndex: 0);
  // Then game's deltaHeroHealth was called with (Peddersen, -1)
  assertThat(spy.lastCall, is( value: "deltaHeroHealth(PEDDERSEN,-1)"));
}
```

# Outlook

- EtaStone CardEffects
  - The core of HearthStone

| Name | Attributes | Effect |
|---|---|---|
| Brown Rice | (1, 1, 1) | Deal 1 damage to opponent hero. |
| Tomato Salad | (2, 2, 2) | Add +1 attack to random minion. |
| Poke Bowl | (3, 2, 3) | Restore +2 health to hero. |
| Noodle Soup | (4, 5, 3) | Draw a card. |
| Spring Rolls | (5, 3, 5) | Destroy a random opponent minion. |
| Baked Salmon | (5, 7, 6) | Add +2 attack to random opponent minion. |

- Really nasty to do Integration Testing
  - *Lots of code to bring card to the field, and test their effects ☹*


# Outlook

- EtaStone
  - Unit testing is *much* easier
    - **Given** poke bowl, **When** executing effect, **Then** game's 'changeHealthOfHero(…)' is called with parameter +2 on my own hero

| Name | Attributes | Effect |
|---|---|---|
| Brown Rice | (1, 1, 1) | Deal 1 damage to opponent hero. |
| Tomato Salad | (2, 2, 2) | Add +1 attack to random minion. |
| Poke Bowl | (3, 2, 3) | Restore +2 health to hero. |
| Noodle Soup | (4, 5, 3) | Draw a card. |
| Spring Rolls | (5, 3, 5) | Destroy a random opponent minion. |
| Baked Salmon | (5, 7, 6) | Add +2 attack to random opponent minion. |