AARHUS UNIVERSITET

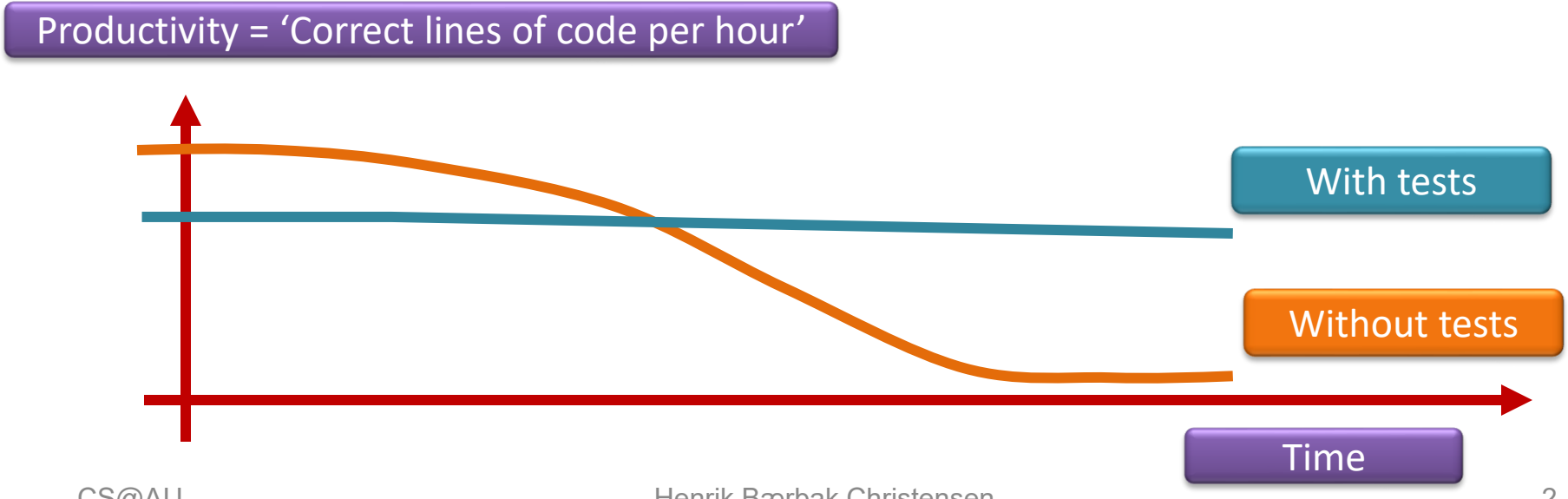
Software Engineering and Architecture

A bit of Motivation and some hints

AARHUS UNIVERSITET

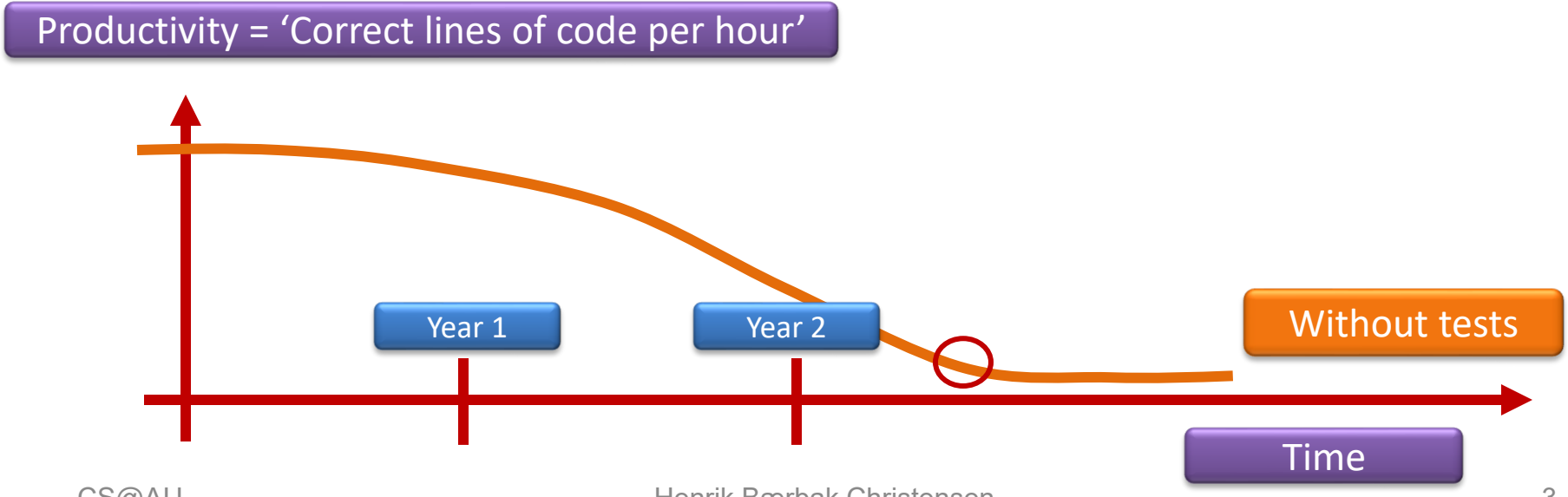
Why TDD?

- *"TDD seems very slow, I need to write all those tests which takes a lot of time, instead of writing production code?"*
- Yeah… But – your productivity keeps nearly constant !



Exercise

- You are employed in a company with little Automated Test tradition… Manual tests…
- What happens here ◯ ? And what happens next?



AARHUS UNIVERSITET

When TDD?

- Do I always do TDD?
  - No! Hacking together a quick Python script to compute SWEA delivery deadlines, or other ‘one-time-tasks’, I just develop using ‘run-and-see-it-work’ (but I always ‘Take Small Steps’)
    - The first running code is always “Hello my new script” printed!
- Testing and TDD is for larger systems with a longer time period…
  - Sometimes ‘one-time-prototypes’ become real systems
- WarStory:
  - That quick-and-dirty prototype, that actually worked…
  - Had to spend 14 days putting in tests to get productivity up again!

AARHUS UNIVERSITET

Why Pair Programming?

- Why waste time on two people doing one person's coding?

- SWEA motivation
  - Get everybody to the keyboard – *training is vital*

- XP motivation
  - Knowledge sharing and mutual learning
  - Keep quality high

AARHUS UNIVERSITET

Mandatory Notes

AARHUS UNIVERSITET

Evident Tests

- The ‘GWT’ comments help a lot, I find…
  - Last year TA’s wanted some guidance so I pulled out ‘TestAlphaStone’

```
@Test
public void shouldRefillManaWhenTurnBegins() {
  // Given Findus plays Tres
  Card tres = game.getCardInHand(Player.FINDUS, 0);
  assertThat(game.playCard(Player.FINDUS, tres), is(Status.OK));
  // And turn goes to opponent
  game.endTurn();
  // Then mana stays at 0
  assertThat(game.getHero(Player.FINDUS).getMana(), is(0));
  // When turn gets back to Findus
  game.endTurn();
  // Then mana is back to 3 (Alpha spec)
  assertThat(game.getHero(Player.FINDUS).getMana(), is(3));
}
```

```
1-2-2022
---

Began development of API and TDD of AlphaStone.

Began 13.00


14.20: hand and field in place; toggle player in turn; initial fielding

15.30: Hero introduced, play Card mana cost deducted from Hero; no
play of cards if not enough mana.

16.15: Attack introduced, health reductions, card removal, validation
of active card (and <? extends Card> changes in API).

Total: 4.5 h
```

AARHUS UNIVERSITET

Incorrect Assumptions

- Producing code means making mistakes, ***also in the test!***
  - Defects often arise when you have *incorrect assumptions*
- Step 1: Quickly Write a Test
  - Given game with Dos, Cuatro in Findus' hand
    - And Dos is at index 0 in the hand
  - When I play Dos, Then the field has 3 cards (Uno+Tres already there)
- Step 2: Run all tests to see the new one fail

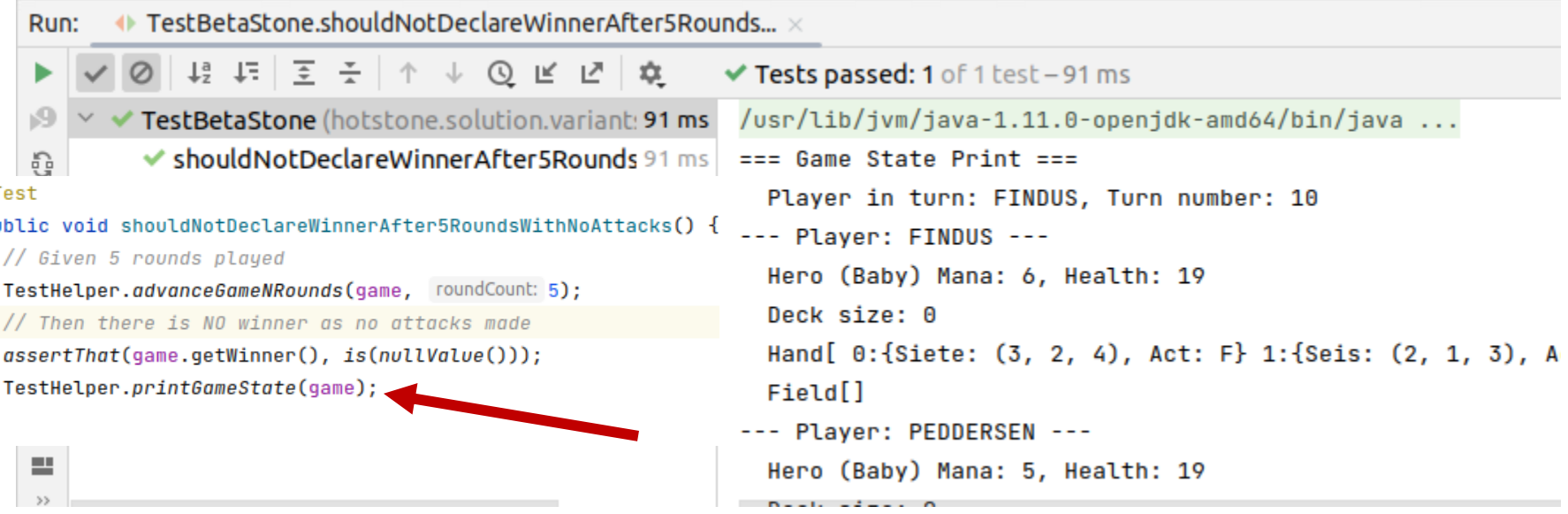
  Red Bar
  - *"But the error reported is some weird null pointer / index out of range error?"*
- ***15 min later: Aah, Dos had already been played !!!***

AARHUS UNIVERSITET

Take Small Steps

- Small steps: *Ensure your assumptions are correct*
  - *Verify that the hand is actually Dos + Cuatro*
- Helper methods to 'dump complete game state'
  - TestHelper.printGameState(game);
  - *Remove the 'printing' once your tests runs correctly!*

Provided in HotStone Code



AARHUS UNIVERSITET

What is TDD???

- Traditional tests = Quality Assurance Technique
  - Success:
    - Tests are constructed to catch defects

- TDD tests = ***Implementation Technique***
  - Success: test cases that *drive implementation*
  - Perhaps a few more to show absence of defects

  - Not a comprehensive quality assurance technique

AARHUS UNIVERSITET

When Do I Stop?

- TDD of the Turn Handling in HotStone
  - Test 1: Given game, Findus is in turn *Fake it*
  - Test 2: Given game, end turn, Pedd. in turn *Triangulation*
- *Exercise – Do I need?*
  - Test 3: Given game, 2x end turn, Findus in turn ?
  - Test 4: 3x end turn, Pedd in turn ?
  - Test 5: 4x end turn, Findus in turn ?
  - Test 17: 16x end turn, Findus in turn ?
- *Exercise – Would it not be clever to do a test like*
  - N from 1..100, Nx end turn; N%2==0 then Findus in turn ?

AARHUS UNIVERSITET

When Do I Stop?

- Add test cases until the particular algorithm is complete and correct **and then stop!**

> TDD Principle: **Representative Data**
>
> What data do you use for your tests? Select a small set of data where each element represents a conceptual aspect or a special computational processing.

- **Overtesting** is harmful
  - Overtesting = same algorithmic production code is tested in numerous different test cases
  - *Exercise: Why is that so, do you think?*
    - Hint: consider that functional requirement changes a bit…

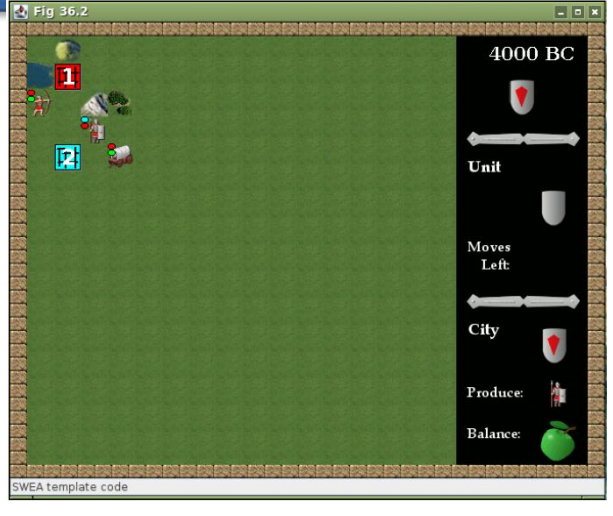
AARHUS UNIVERSITET

Doing TDD

- **Evident Tests –** *make tests easy to understand*

```
@Test
  public void plainsEveryWhereBut2_2And1_0And0_1()
  {
     //Iterate through all tiles in world
     for(int row = GameConstants.WORLDSIZE-1; row>=0 ; row--) {
         for(int column = GameConstants.WORLDSIZE-1; column >= 0 ; column--) {
             if(!(row == 1 && column == 0 || row == 2 && column == 2 || row == 0 && column == 1)) {
                 assertEquals("There should be plains at " + row + "," + column,
                     game.getTileAt(new Position(row,column)).getTypeString(), GameConstants.PLAINS);
             }
         }
     }
  }
```

Sorry – AlphaCiv example…



- Exercise: What is focus here?
  - Test that everything works? Or
  - Drive production code into existence?
- And – is it Evident ?

AARHUS UNIVERSITET

Stable Test cases

- The *more* your testcases *only* use the given Game, Card, Hand interfaces…
- The more *stable* your test cases will be against refactoring/changing inner data structures!

- So
  - game.getCardInHand(FINDUS, 1) ☺
- Never, ever things like
  - ((GameImpl) game).internalHandArray[0][1] ☹
- Exercise: ***Why not?***

AARHUS UNIVERSITET

Design Issues

- *Which data structure should I use?*
  - Anyone you like
    - Arrays — Card[]
    - Lists — List<Card>
    - Maps — Map<Player, Hero>

I recommend, however, to avoid raw Java Arrays. Use the collection libraries.

- As there 'is two of everything' you will likely combine
  - List of arrays, or maps of lists, or arrays of lists, or array of arrays, or … Ala *Map<Player, List<Card>>*
- If using arrays, remember 'ordinal()' of an Enum
  - Player.FINDUS.ordinal() == 0

Do the same thing, the same way

Mutation

Who has access to mutation?

Answer: Only Game, it handles the rules of the game…

AARHUS UNIVERSITET

SideBar: For Python People

- Java is a type-safe language, meaning **types are checked**
  - If an object is of type 'Card' only the methods mentioned in that type can be called
    - If you call other methods; the compiler will refuse!
    - (Even if the underlying object really *does* implement that method.)

```
interface Card {
  int getHealth();
}
class StandardCard implements Card {
  int h;
  int getHealth() { return h; }
  void setHealth(int v) { h = v; }
}

Card c = game.getCard...(...);
int h = c.getHealth();  // OK
c.setHealth(7);         // NOT OK

StandardCard sc = game.getCard...(...);
int h = sc.getHealth();  // OK
sc.setHealth(7);         // OK

// Casting...
StandardCard sc = (StandardCard) c;  // may fail
sc.setHealth(7);         // OK
```

AARHUS UNIVERSITET

Those ‘read-only’ interfaces

- I stated that *try to keep Card, Hero as read only interfaces*
  - *That is, they only have accessor methods, no mutator methods*
    - *Only ‘getX()’, never a ‘setX(int newValue)’*
- *Why?*
  - Actually it is the ‘Facade’ pattern which we will return to later, but
  - Main point:
    - game.getHero(Peddersen).addToHealth(1000);
    - *is not obeying the rules of the game and must be guarded against!*
- How?
  - (next slide, please)

AARHUS UNIVERSITET

Mutating Internal State

- Card and Hero should be *read-only interfaces*
- Then how can Game every change, say, hero mana left?
- Solution for now:
  - A) Add mutators to the **implementing** classes
    - StandardHero::reduceManaLeft(int byValue)
  - B) In Game *either*
    - Declare by concrete type
      - List<StandardHero> theHeros;
    - Or, Use casts when needed
      - StandardHero hero = (StandardHero) getHero(FINDUS);
- Why is this OK?
  - Well – Game is responsible for mutations and know concrete type

Actually…

- … We can find a better solution for this
  - "Private interfaces"

- We will come back to this point later…

- *SWEA is a course where it is good to know everything in advance*
  - *But that is not how learning works, right?*