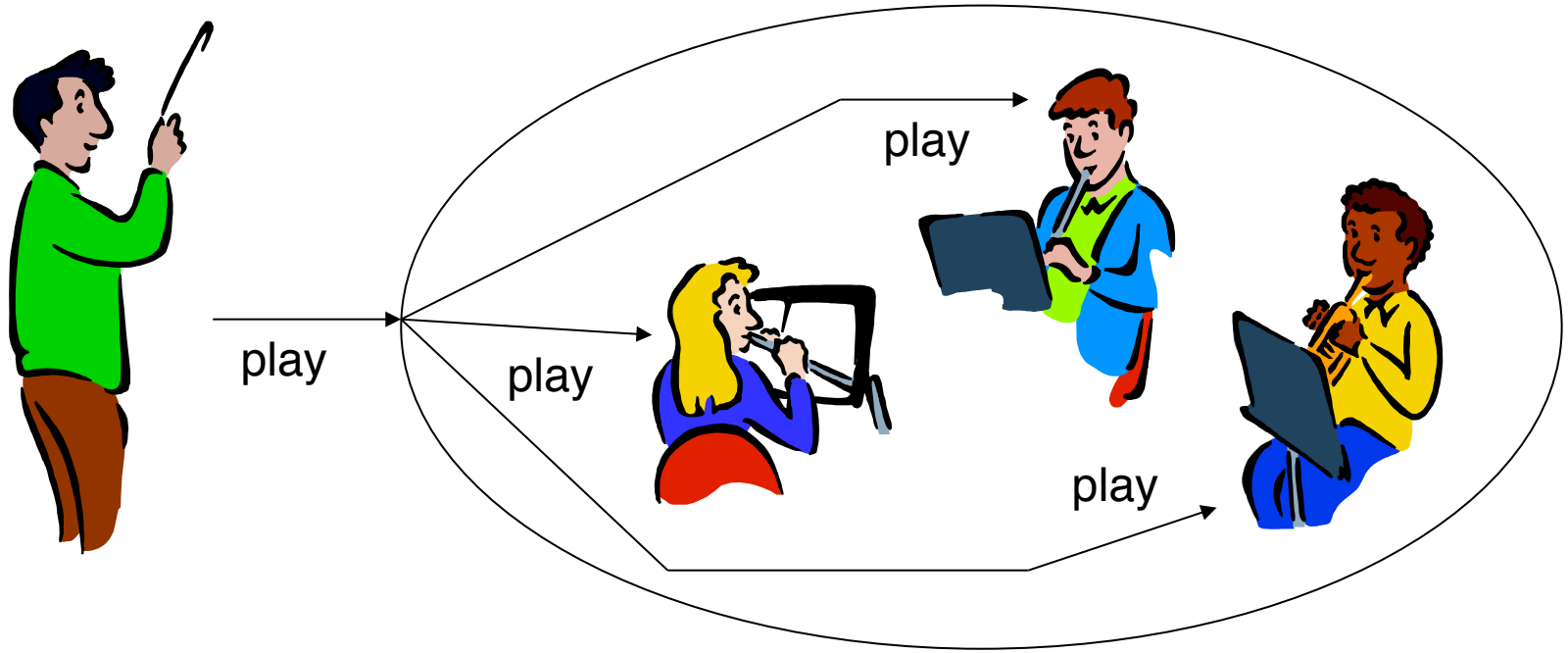
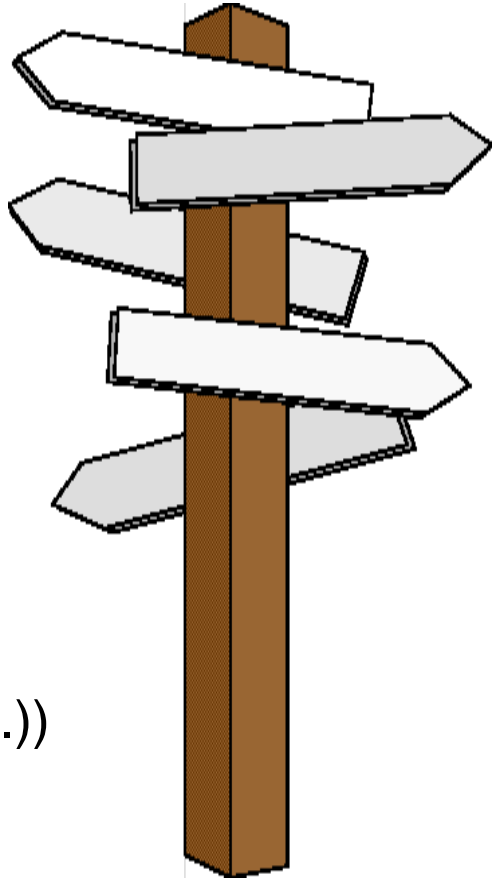
Complexe Interacties

Een optimale werkverdeling

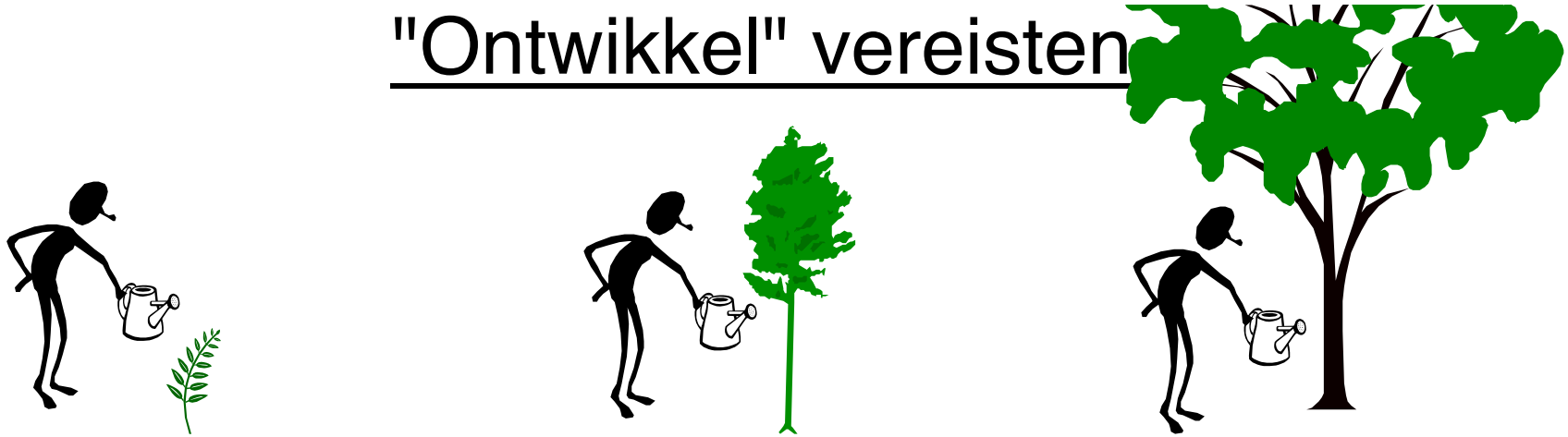


3. Aanpasbaarheid

- Aanpasbaarheid: TicTacToe
  - versie 1.1b (displayGame)
    - koppeling / cohesie
  - versie 1.2 (Player)
    - domeinmodel
  - versie 1.3 (Player.moves)
    - vermijd gebruikersinvoer
  - versie 1.4 (Player. winner())
    - Basisfunctionaliteit
  - versie 1.5 (TictacToeTest.compareFiles(...))
    - ASCII uitvoer
- Conclusie



"Ontwikkel" vereisten



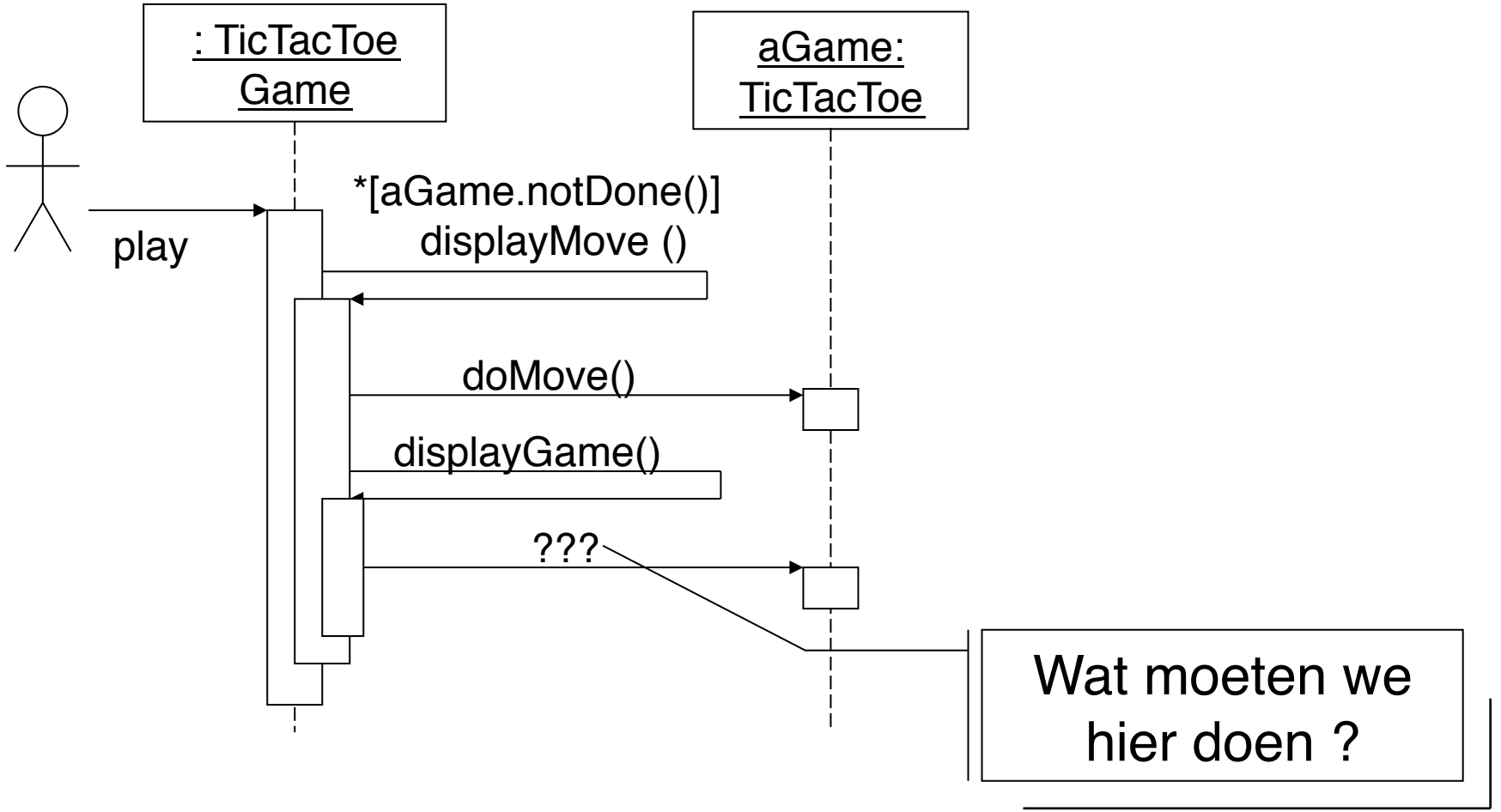
Vereisten

- Betrouwbaarheid
- Aanpasbaarheid
- Planning

Technieken

- Testen + Contracten
- Objectgericht ontwerp
- Tijdsschatting

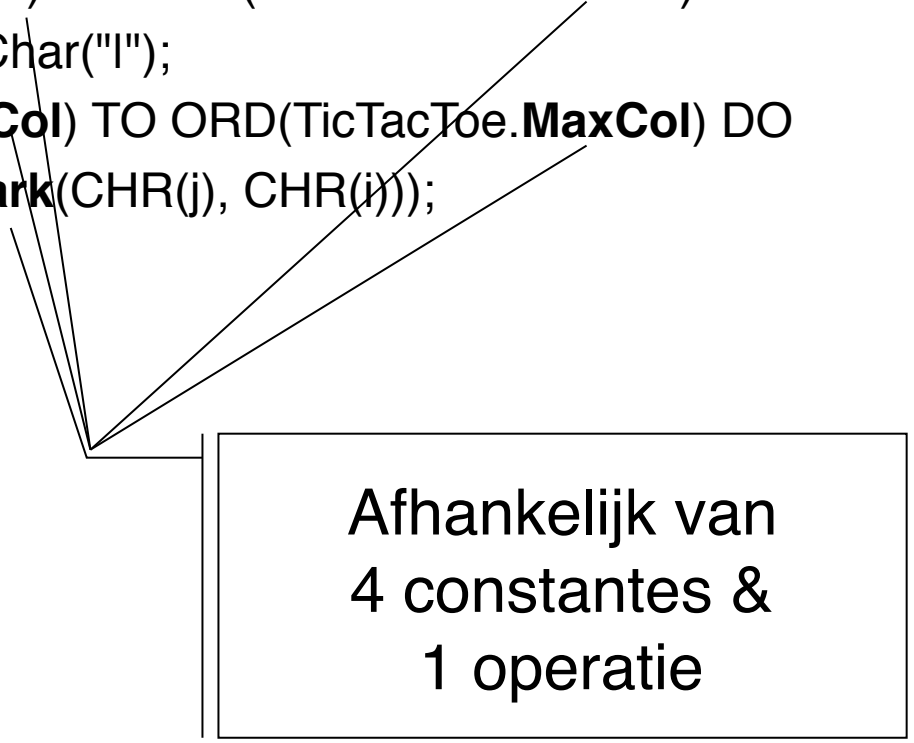
TicTacToeGame (1.1b)



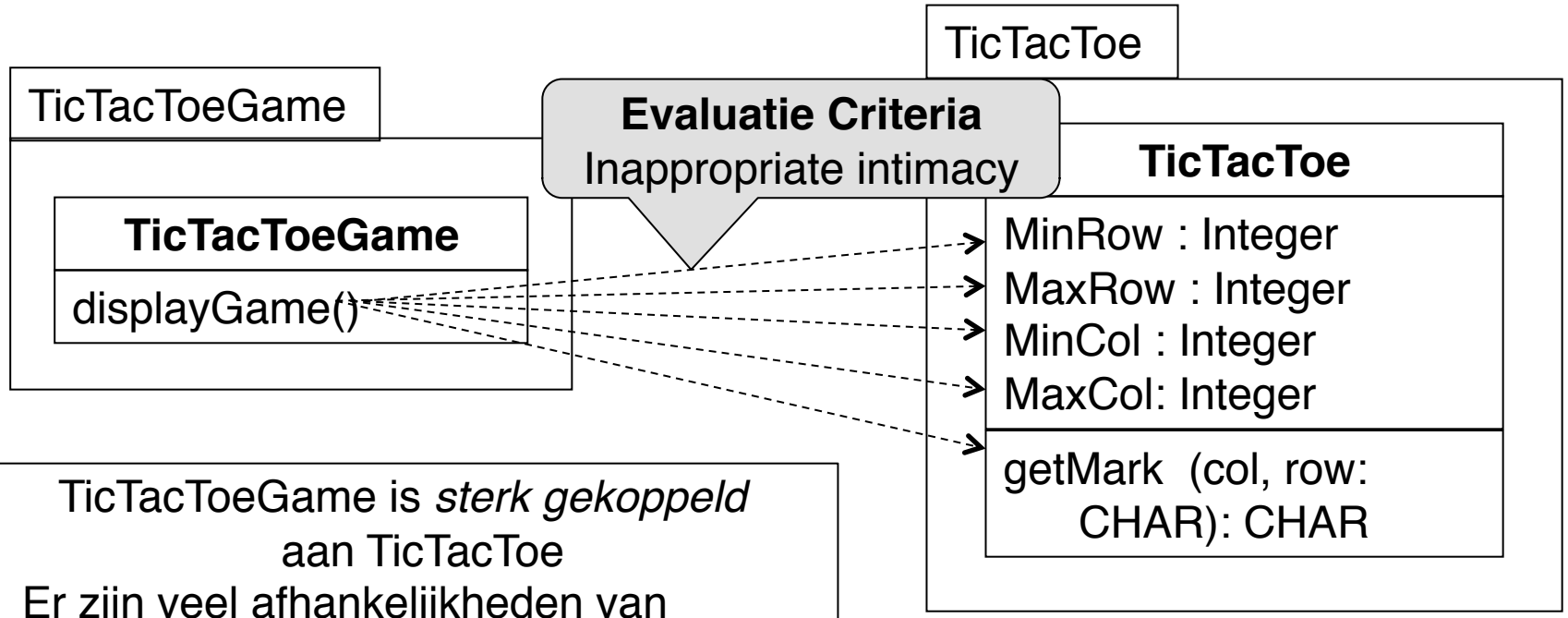
```
PROCEDURE Play*;
   VAR aGame: TicTacToe.TicTacToe;
BEGIN
   initDisplay;
   NEW(aGame);
   aGame.init();
   displayGame(aGame);
   WHILE aGame.notDone() DO
      aGame.doMove();
      displayGame(aGame);
   END;
END Play;
```

Wat moeten we hier doen ?

```
PROCEDURE displayGame(aGame: TicTacToe.TicTacToe);
   VAR i, j: INTEGER;
BEGIN
   ...
   FOR i := ORD(TicTacToe.MinRow) TO ORD(TicTacToe.MaxRow) DO
      OutExt.Char(CHR(i));OutExt.Char("|");
      FOR j := ORD(TicTacToe.MinCol) TO ORD(TicTacToe.MaxCol) DO
         OutExt.Char(aGame.getMark(CHR(j), CHR(i)));
         OutExt.Char("|");
      END;
      OutExt.Ln;
      ...
   END;
   OutExt.Ln;
END displayGame;
```



Sterke Koppeling



TicTacToeGame is *sterk gekoppeld* aan TicTacToe

- Er zijn veel afhankelijkheden van TicTacToeGame naar TicTacToe
- Een verandering aan TicTacToe betekent meestal dat we ook TicTacToeGame moeten aanpassen

Aanpasbaarheid :(

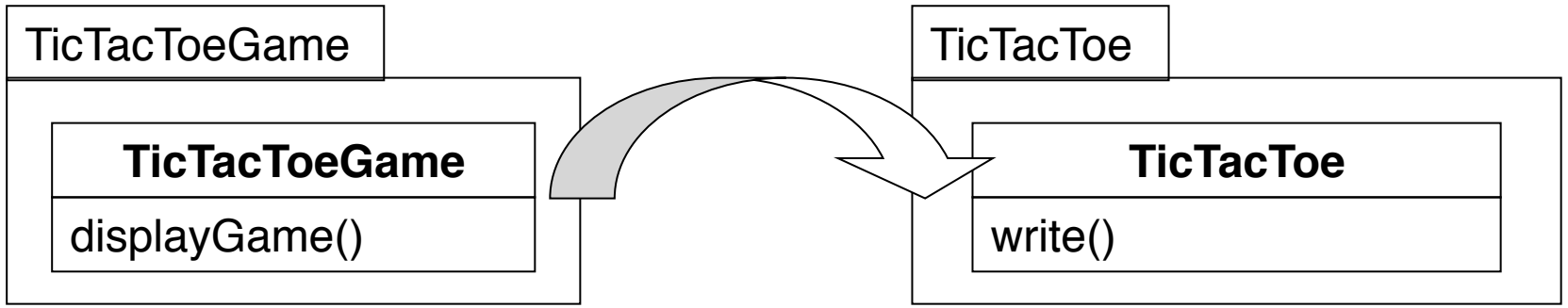
Vuistregel

Plaats het gedrag
dicht bij de data

Refactor:

- Routines die veel "get" operaties oproepen
  => verplaats naar de corresponderende klasse.

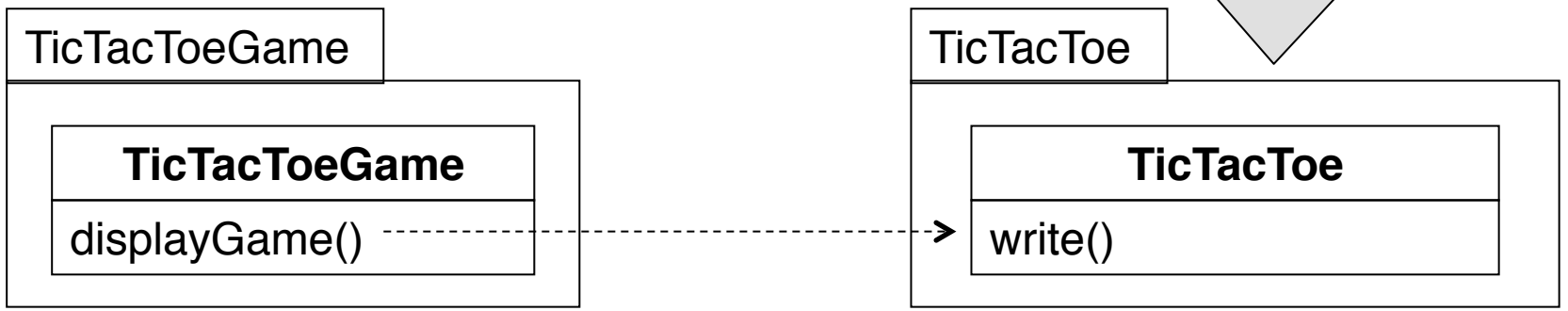
Plaats gedrag bij Data




```
FOR i := ORD(TicTacToe.MinRow) TO
            ORD(TicTacToe.MaxRow) DO
  OutExt.Char(CHR(i));OutExt.Char("|");
  FOR j := ORD(TicTacToe.MinCol) TO
            ORD(TicTacToe.MaxCol) DO
    OutExt.Char(aGame.getMark(…
```

Zwakke Koppeling

Evaluatie Criteria
Data Encapsulatie



TicTacToeGame is *zwak gekoppeld* aan TicTacToe

- Er is maar één afhankelijkheid van TicTacToeGame naar TicTacToe
- Een verandering aan TicTacToe heeft zelden een effect op TicTacToeGame

Aanpasbaarheid :)

```
PROCEDURE displayGame(aGame: TicTacToe.TicTacToe);
   BEGIN
      aGame.write;
   END displayGame;
```

Eén afhankelijkheid van TicTacToeGame naar TicTacToe

```
PROCEDURE (aTicTacToe: TicTacToe) write* ();
         VAR i, j: INTEGER;
   BEGIN
      OutExt....
      FOR i := ORD(MinRow) TO ORD(MaxRow) DO
         OutExt.Char(CHR(i));OutExt.Char("|");
         FOR j := ORD(MinCol) TO ORD(MaxCol) DO
            OutExt....
         END;
         ...
```

Maar nu is TicTacToe afhankelijk van OutExt
=> Geen uitvoer op Out, Oberon.Log

Vuistregel

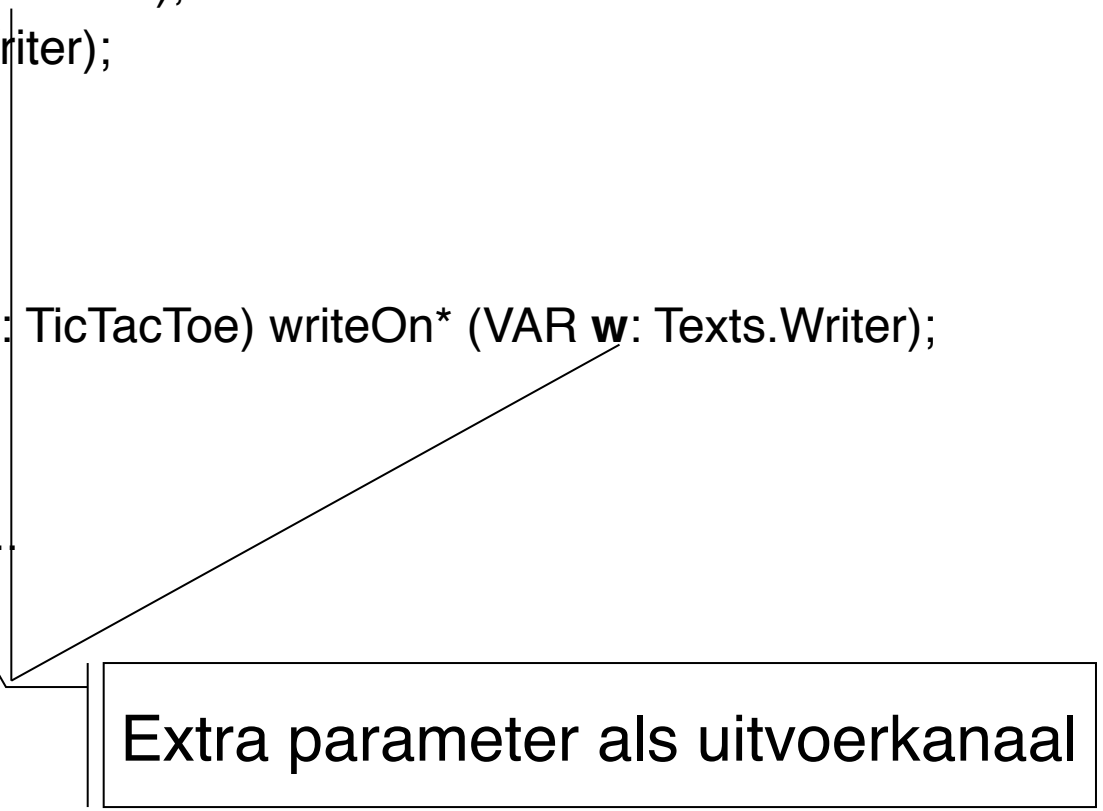
Nooit user-interface code
in de basisklassen

Refactor:

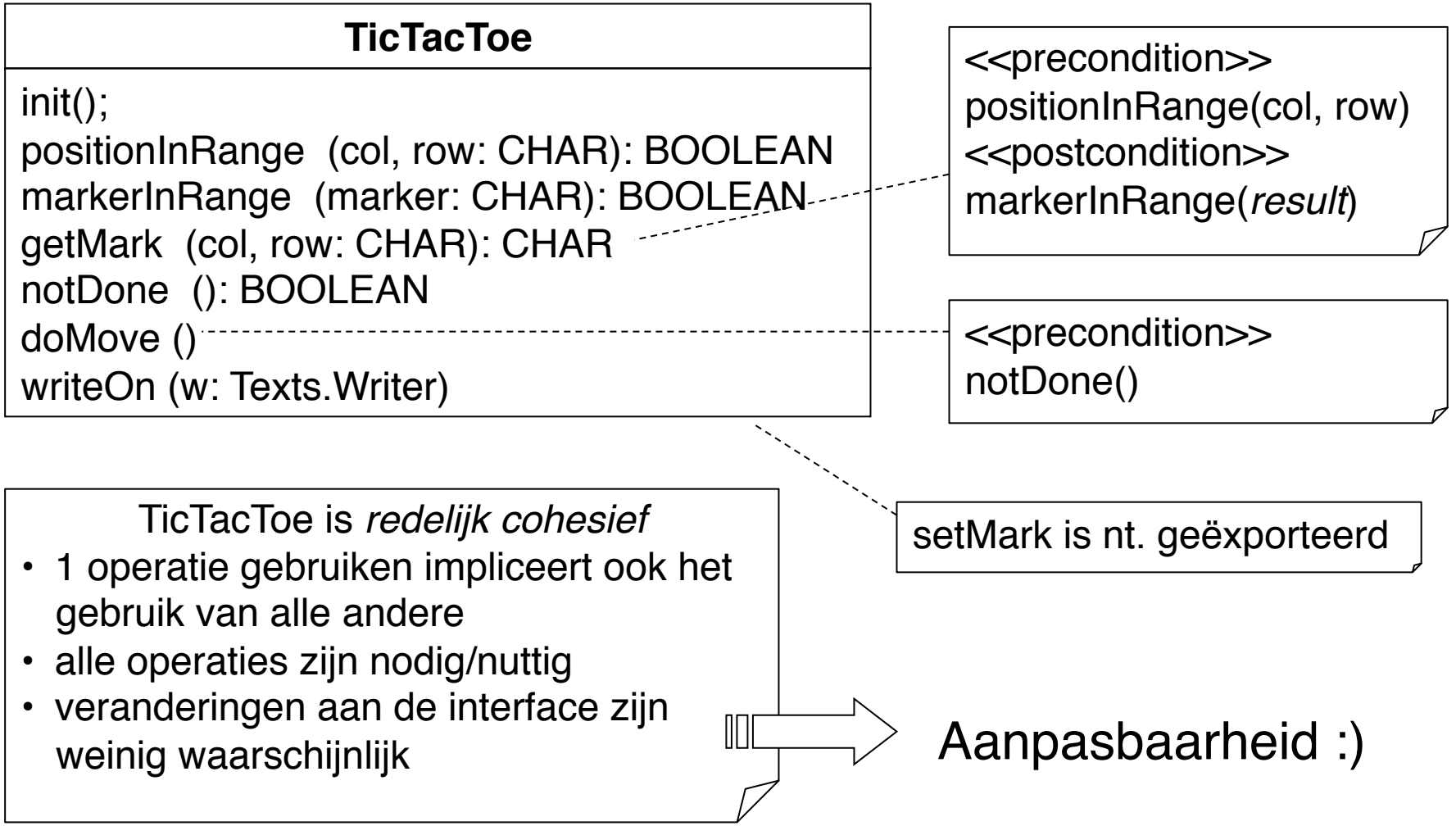
- Extra parameter als in/uitvoerkanaal naar de buitenwereld

```
PROCEDURE displayGame(aGame: TicTacToe.TicTacToe);
   BEGIN
      aGame.writeOn(globalWriter);
      OutExt.Writer(globalWriter);
   END displayGame;


PROCEDURE (aTicTacToe: TicTacToe) writeOn* (VAR w: Texts.Writer);
         VAR i, j: INTEGER;
   BEGIN
      Texts.WriteString(w, ....
      ...
```



Cohesie


TicTacToe
init();
positionInRange (col, row: CHAR): BOOLEAN
markerInRange (marker: CHAR): BOOLEAN
getMark (col, row: CHAR): CHAR
notDone (): BOOLEAN
doMove ()
writeOn (w: Texts.Writer)
<<precondition>>
positionInRange(col, row)
<<postcondition>>
markerInRange(result)
<<precondition>>
notDone()
setMark is nt. geëxporteerd
TicTacToe is redelijk cohesief
• 1 operatie gebruiken impliceert ook het gebruik van alle andere
• alle operaties zijn nodig/nuttig
• veranderingen aan de interface zijn weinig waarschijnlijk
Aanpasbaarheid :)

Koppeling vs. Cohesie

Koppeling

= mate waarin een component afhankelijk is van andere componenten

- te MINIMALISEREN
  => veranderingen hebben minder impact

Cohesie

= mate waarin de onderdelen van een component afhankelijk zijn van elkaar

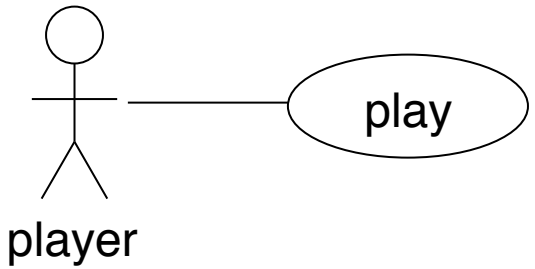
- te MAXIMALISEREN
  => veranderingen zijn minder waarschijnlijk

Ideaal

= een component die niks doet

=> perfectie is niet haalbaar

TTT1.2



Use Case 1: play

- Goal: 2 players play TicTacToe, 1 should win
- Precondition: An empty 3x3 board
- Success end: 1 player is the winner

Steps

1. Two players start up a game (First is "O"; other is "X")
2. WHILE game not done
   - 2.1 Current player makes move
   - 2.2 Switch current player
3. Anounce winner

Waar voegen we dit bij ?

Vuistregel

Evaluatie Criteria
Goeie ADT

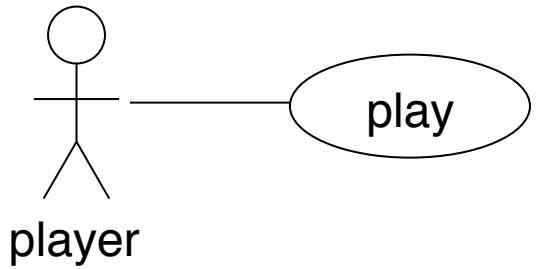
Maak een model van het probleemdomein

= het *DOMEINMODEL*

Tips:

- Zelfstandige naamwoord als indicator voor object / klasse.
- Werkwoord als indicator voor operatie
- Naamgeving in basisklassen = naamgeving in probleemdomein

Zelfstandige Naamwoorden



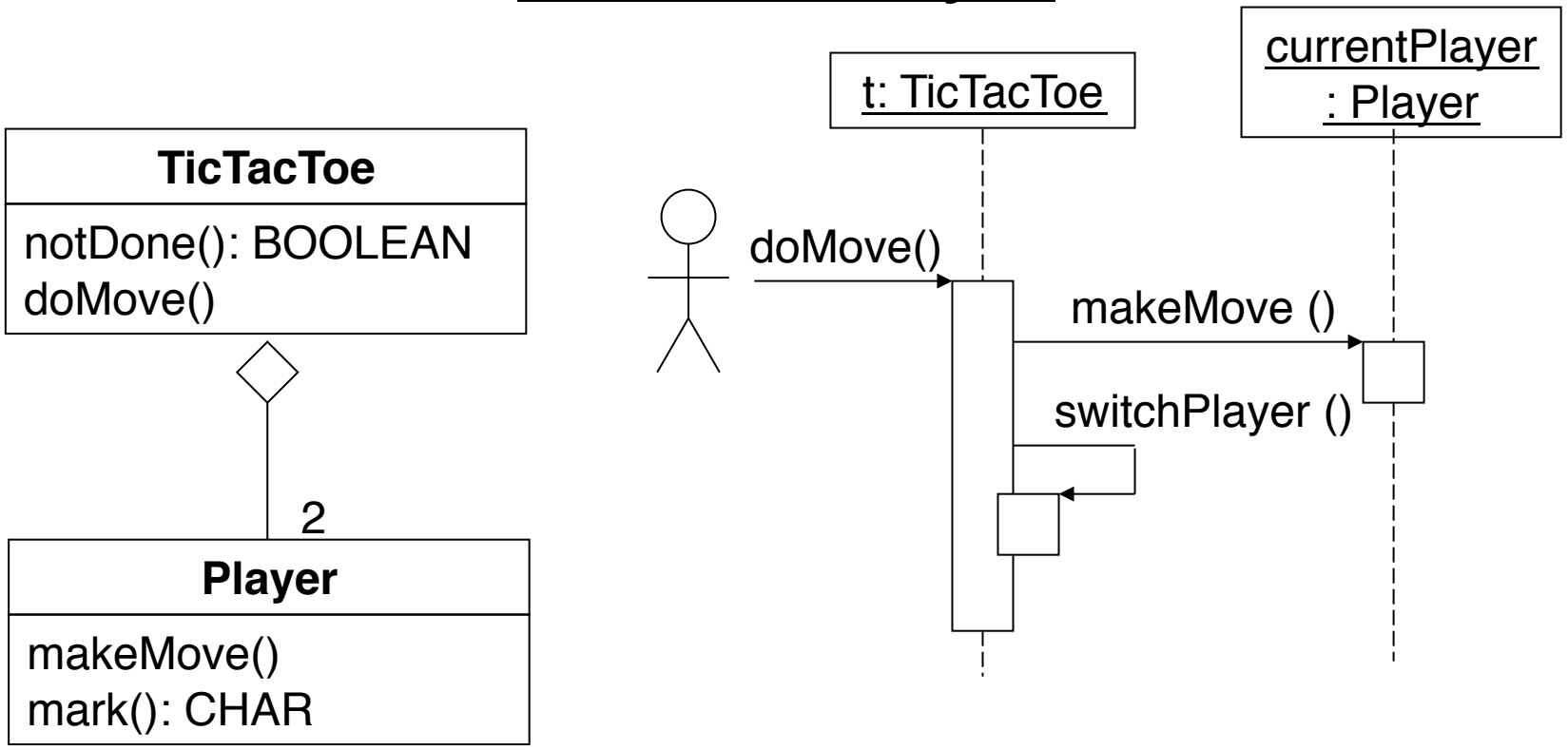
Legende
- Substantief
- *Werkwoord*

Use Case 1: play
- Goal: 2 players play TicTacToe, 1 should win
- Precondition: An empty 3x3 board
- Success end: 1 player is the winner

Steps

1. Two players *start up* a game
   (First is "O"; other is "X")
2. WHILE game not done
   2.1 Current player *makes move*
   2.2 *Switch* current player
3. *Anounce* winner

TTT1.2: Player



```
PROCEDURE (aTicTacToe: TicTacToe) doMove* ();
   VAR row, col, mark: CHAR;
   BEGIN
      ASSERT(aTicTacToe.notDone(), 100);
      IF ODD(aTicTacToe.nrOfMoves) THEN
          mark := "X" ELSE mark := "O"; END;

      col := CHR((aTicTacToe.nrOfMoves MOD 3) + ORD("a"));
      row:= CHR((aTicTacToe.nrOfMoves DIV 3) + ORD("1"));
      aTicTacToe.setMark(col, row, mark);

      aTicTacToe.nrOfMoves := aTicTacToe.nrOfMoves + 1;
END doMove;
```

Verplaats dit stuk code naar Player

```
TYPE
   Player* = POINTER TO PlayerData;
   PlayerData* = RECORD
      marker: CHAR;
   END;
...
PROCEDURE (aPlayer: Player) initMarked* (m: CHAR);
...
PROCEDURE (aPlayer: Player) makeMove*  (aTicTacToe: TicTacToe);
   VAR row, col: CHAR;
   BEGIN
      ASSERT(aTicTacToe.notDone(), 100);
      col := CHR((aTicTacToe.nrOfMoves MOD 3) + ORD("a"));
      row:= CHR((aTicTacToe.nrOfMoves DIV 3) + ORD("1"));
      aTicTacToe.setMark(col, row, aPlayer.mark());
   END makeMove;
```

Verplaatste code ...

```
TYPE
   TicTacToeData* = RECORD
      ...
      players: ARRAY 2 OF Player;
   END;
PROCEDURE (aTicTacToe: TicTacToe) init*;
   ...
   NEW(aTicTacToe.players[0]); NEW(aTicTacToe.players[1]);
   aTicTacToe.players[0].initMarked("O"); aTicTacToe.players[1].initMarked("X");
END init;
...
PROCEDURE (aTicTacToe: TicTacToe) doMove* ();
BEGIN
   ASSERT(aTicTacToe.notDone(), 100);
   aTicTacToe.players[aTicTacToe.nrOfMoves MOD 2].makeMove(aTicTacToe);
   aTicTacToe.nrOfMoves := aTicTacToe.nrOfMoves + 1;
END doMove;
```

... vervangen door operatie

Vuistregel

Een test verwacht ***géén*** gebruikersinvoer

Waarom ?

- Veel & frequent testen => onmogelijk om steeds invoer te geven

Hoe dan wel ?

- Invoerdata wordtgecreëerd door testprogramma (testbestand ?)
- Ontwerp basisklassen onafhankelijk van het data invoer kanaal

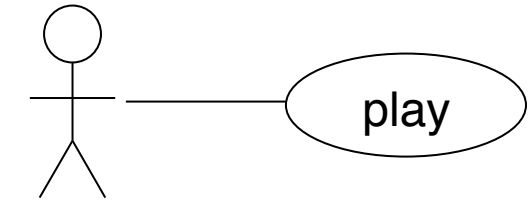
TTT1.3: Reeks van zetten

```
PROCEDURE TicTacToeTest.Main;
   ...
BEGIN
   NEW(aTest); aTest.init();
   aTest.setUp("testBasicGame",
      "a1 c1 b2 a3 c3", "b1 a2 c2 b3");
   IF ~ aTest.testBasicGame (TRUE) THEN testsPassed := FALSE; END;
   aTest.tearDown();
   ...
END Main;
...
PROCEDURE (aTicTacToe: TicTacToe) init* (mvO, mvX: ARRAY OF CHAR);
...
PROCEDURE (aPlayer: Player) initMarkedMoves* (marked: CHAR; moves:
   ARRAY OF CHAR);
```

	a	b	c
1	O	X	O
2	X	O	X
3	O	X	O

een reeks zetten
=> invoerdata door testprogramma

Uitzonderingen



Use Case 1: play

- …

Steps

1. Two players start up a game
   (First is "O"; other is "X")
2. WHILE game not done
   - 2.1 Current player makes move
   - 2.2 Switch current player
3. Anounce winner

Exceptions

2.1. [Illegal Move] System issues a warning
   => continue from step 2.1

tijdens 2.1 kan "Illegal Move" voorkomen

Vuistregel

Minstens één test per "uitzondering"

Waarom ?

- Alle scenarios in de specificaties moeten getest worden

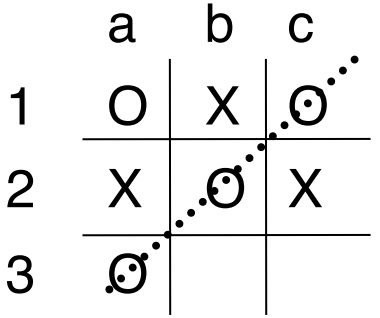
Hoe ?

- Controleer resultaat uitzondering via "should" & "shouldNot"
  => denk eraan: " Een test produceert zo weinig mogelijk uitvoer"

```
PROCEDURE (t: TicTacToeTest) testLegalMoves* (v: BOOLEAN): BOOLEAN;
    PROCEDURE aux (in: ARRAY OF CHAR; legal: BOOLEAN): BOOLEAN;
    VAR result: BOOLEAN;
    BEGIN
        result := TicTacToe.LegalMoves(in);
        IF ~ aTest.should(legal = result, …) THEN RETURN FALSE; END;
    END
  BEGIN
    IF ~ aux("a1 c1 b2 a3 c3", TRUE) THEN RETURN FALSE END;
    IF ~ aux("b1 a2 c2 b3", TRUE) THEN RETURN FALSE END;
    IF ~ aux("", TRUE) THEN RETURN FALSE END;
    IF ~ aux("    b1 a2 c2 b3    ", TRUE) THEN RETURN FALSE END;
    IF ~ aux("A1", FALSE) THEN RETURN FALSE END;
    IF ~ aux("a5", FALSE) THEN RETURN FALSE END;
    IF ~ aux("a19", FALSE) THEN RETURN FALSE END;
    RETURN TRUE;
  END testLegalMoves;
```

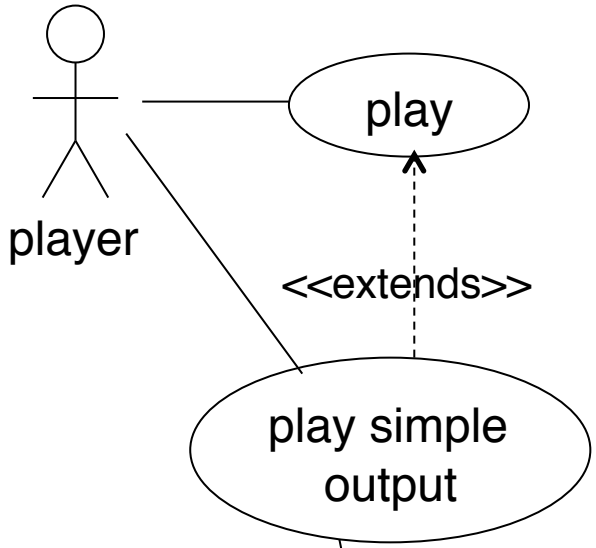
TTT1.4: Winnaar

```
PROCEDURE TicTacToeTest.Main;
   ...
BEGIN
   NEW(aTest); aTest.init();
   aTest.setUp("testRealGame (diagonal wins)",
         "a1 c1 b2 a3 c3", "b1 a2 c2 b3");
   IF ~ aTest.testRealGame (TRUE, "O", 7) THEN testsPassed := FALSE; END;
   aTest.tearDown();
   ...
END Main;
```



Test verifieert winnaar
+ aantal zetten

ASCII Uitvoer (TTT15)



Use Case 2: play simple output

- Extension of use case 1

Steps

use case 1 + additional steps

afer 2.2

2.3 write game on ASCII text

during 3

3 write winner on ASCII text

(Minstens één test per "normaal" scenario)
nieuwe use case => uitbreiding tests

```
PROCEDURE (aTest: TicTacToeTest) testOutputGame* (verbose: BOOLEAN;
      useOutput,  expectedOutput: ARRAY OF CHAR): BOOLEAN;
…
   Texts.OpenWriter(w);
   WHILE aTest.aGame.notDone() DO
      aTest.aGame.doMove();
      aTest.aGame.writeOn(w);
   END;
   lenWritten :=aTest.writerToFile(w, useOutput);
   IF ~ aTest.should(aTest.compareFiles(useOutput, expectedOutput),
         "output written is not what was expected")
…
END testOutputGame;
…
aTest.setUp("testOutputGame (full board)", "a1 c1 b2 b3 c3", "b1 a2 c2 a3");
IF ~ aTest.testOutputGame (FALSE, "testOutput1.txt", "fullboard.txt")
```

Vergelijk gegenereerde met verwachte uitvoer

oproep v.d. test

Vuistregel

Test lange uitvoer door het vergelijken van files.

Waarom ?

- Veel & frequent testen => makkellijk om steeds uitvoer te testen

Hoe ?

- Verwachte uitvoerdata wordt één maal manueel gecreëerd
- Achteraf wordt verwachte uitvoer vergeleken met gegenereerde uitvoer

Conclusie

- Eerst moet je het *doen*
  - KISS principle: klein beginnen en langzaam groeien
  - tests lopen + contracten worden nageleefd
- Daarna moet je het *goed* doen
  - lage *koppeling*, hoge *cohesie*
  - *model* van het probleemdomein

} goed ontwerp

⇒en ... tests blijven lopen
+ contracten blijven nageleefd

Vuistregels

Ontwerpen

- Plaats het gedrag dicht bij de data
- Nooit user-interface code in de basisklassen
- Maak een model van het probleemdomein (= het domeinmodel)
  - zelfstandige naamwoorden & werkwoorden als indicators

Testen

- Een test verwacht ***géén*** gebruikersinvoer
- Minstens één test per "uitzondering"
- Test lange uitvoer door het vergelijken van files