

Project Software Engineering

Cursus

1ste Bachelor Informatica
Academiejaar 2015-2016

© Serge Demeyer – Universiteit Antwerpen



Universiteit Antwerpen

1. Praktisch



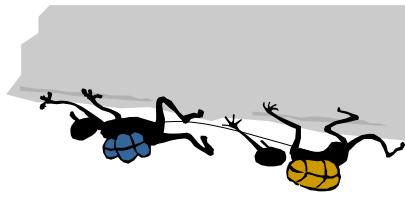
- Doel
- Contactpersonen
- Inhoud
- Opbouw
- Mijlpalen
- Tijdsbesteding
- Eindbeoordeling
- Spelregels (=> Fraude)
- Cursusmateriaal

<http://ansymore.uantwerpen.be/courses/SE1BAC>

1. Praktisch

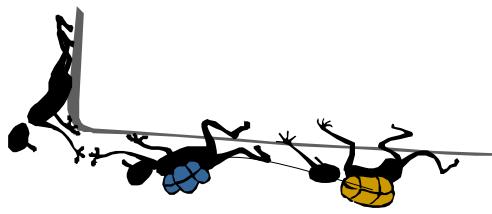
Doel

- Een eerste ervaring verwerven ...
 - doen, niet kijken hoe het gedaan wordt
 - in het zelf realiseren ...
 - zelf doen ... weliswaar in groepjes van 2
 - van een software oplossing ...
 - informatica doen: jullie interesse voor een niet-triviaal probleem ...
 - een onderdeel van een groter geheel



2

Contactpersonen



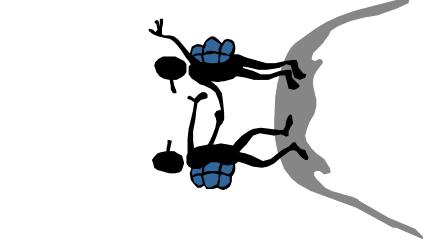
1. Praktisch

Docent

- Serge Demeyer
(Middelheim - G 3de verdieping)
 - serge.demeyer@uantwerpen.be
 - GEEN bijlagen in e-mail

Assistenten

- (Middelheim - G 3de verdieping)
 - Vragen tijdens uren practicum
 - practica zullen gereserveerd !
 - ⇒ praktica zullen gereserveerd !



Project

- Zelfwerkzaamheid
- Duo: in groepjes van 2
 - + Groepen zijn vrij te kiezen ...
 - ... binnen categorieën bepaald door "Programmeren I"
 - + Elk krijgt dezelfde punten

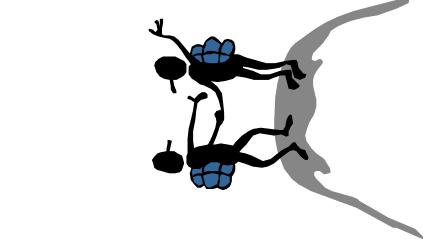
- ... op basis van theorie

- 3-tal lessen

- Minimum toe te passen technieken
 - + testen + contracten
 - + objectgericht ontwerpen
 - + plannen

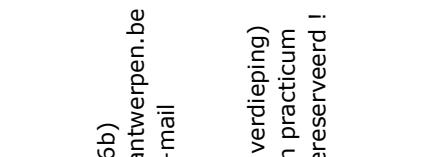
Inhoud

1. Praktisch

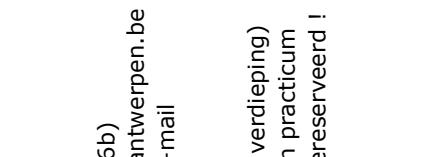


Tijdsbesteding

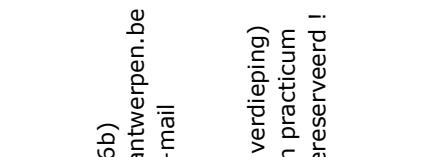
1



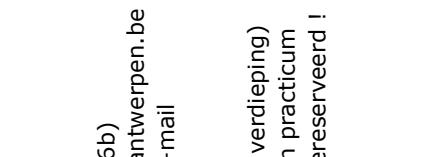
Eindbeoordeling



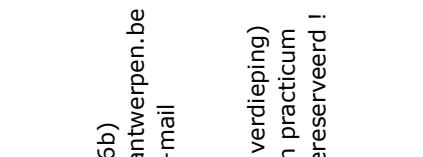
2



Spelregels (=> Fraude)



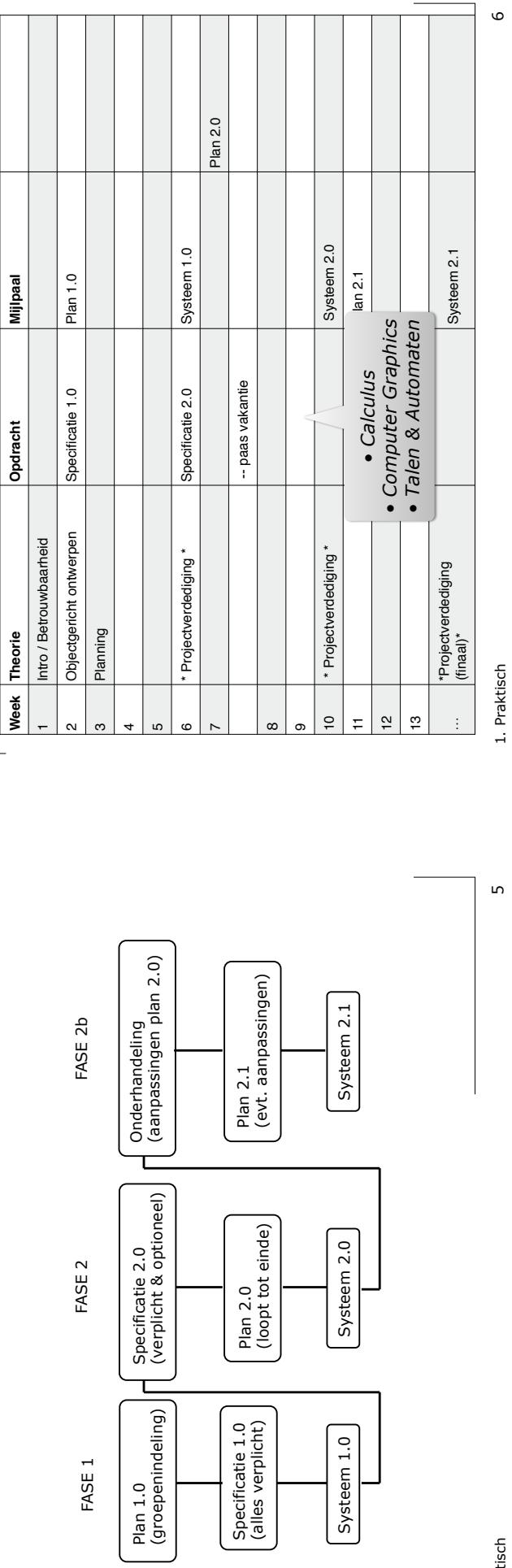
3



4

Oppboww

Semester Indeling (ideale omstændigheden)



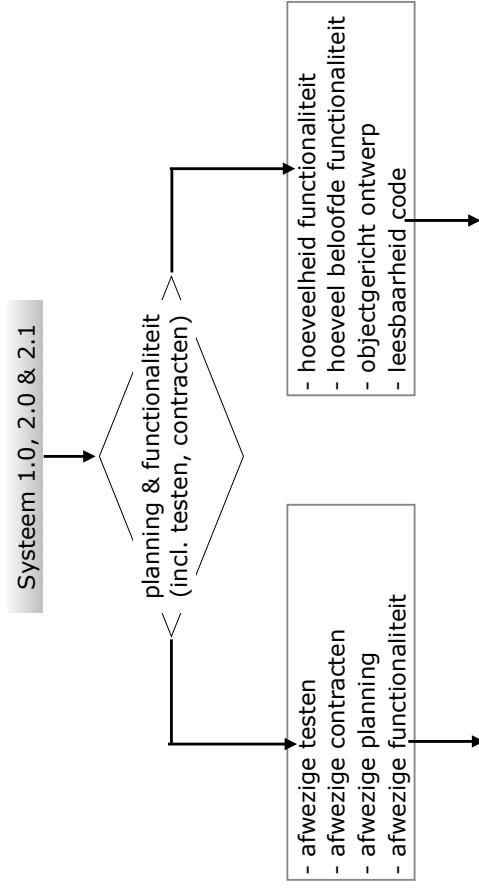
Tijdsbesteding

15u theorie	x 2	$\Rightarrow 30u$ extra studietijd	30	
60u oefeningen	x 1.5	$\Rightarrow 90u$ extra studietijd	150	
TOTAAL		\Rightarrow "werktaid"	180	
TOTAAL per week		$\Rightarrow 180 u \div (13 - 1)$ weken		$\Rightarrow 15 u$ per week (*)

Log de tijd besteed aan je project op speciale tijdsbladen.
⇒ zelfcontrole

(*) Dat is wat wij ongeveer verwachten; niet noodzakelijk wat jullie effectief besteden

Eindbeoordeling

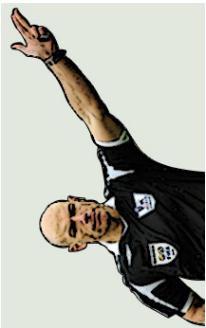


Eindresultaat [0 ... 10[

Eindresultaat [10 ... 20]

Spelregels

- werkt in computerlabo
+ "op de PC thuis werkt het" :(
- stipt opleveren
+ te laat = niet opgeleverd
- accurate tijdsbladen
+ plannen / onderhandelen :)
- maak reservekopieën
+ "gisteren werkte het nog" :(
- werk nauw samen
+ per twee voor één scherm :)
 - 1 programmeur / 1 denker
 - verwijsel de rollen
- meld problemen tijdig
+ partner werkt niet mee
- verboden werk te kopiëren
+ betrapt ⇒ fraude
- bibliotheken (o.a. graphics)
+ zijn toegelaten



Fraude ?



Examenform = Project

- + programma code kopiëren / bekijken ≈ Afkijken
 - + programma code doorgeven ≈ laten Afkijken
- Controle
- + na elke evaluatie vergelijken we alle projecten
 - we houden de historiek bij
- Sanctie
- + uitgesloten voor (een gedeelte van) het vak
 - + uitgesloten voor de zittijd
 - + uitgesloten voor het academiejaar

1. Praktisch

9

1. Praktisch

10

Cursusmateriaal

- Kopies van de transparanten + alle andere informatie
- <http://ansymore.uantwerpen.be/courses/SE1BAC>
 - Zie ook Blackboard
 - volg de aankondelingen + lees je UA e-mail



Engelstalige versie

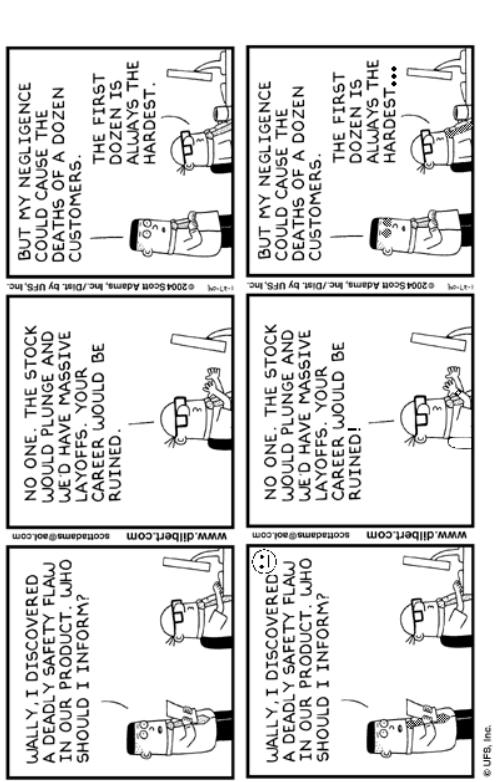
- [Stev99a] Using UML: Software Engineering with Objects and Components, P. Stevens, R. Pooley, Addison-Wesley, 1999.
- [Stev00a] Toepassing van UML: Software-engineering met objecten en componenten, P. Stevens, R. Pooley, Addison-Wesley, 2000.
- Nederlandstalige versie

1. Praktisch

11

Ontdek de 7 verschillen

Een oplossing met een foutje ...



2. Betrouwbaarheid

2. Betrouwbaarheid

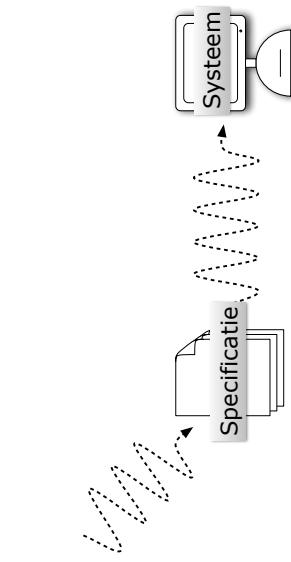
Software Engineering

- Wat?
 - Bouw vs. Ontwikkel
 - Vereisten
 - + Betrouwbaarheid + (Aanpasbaarheid & Planning)
- Betrouwbaarheid: TicTactoe
- specificatie
 - TicTactoe10: ontwerp, implementatie & test
 - TicTactoe11: ontwerp, implementatie met slechte test (manuele controle)
 - TicTactoe12: contracten & eerste demonstratie

Wat is Software Engineering ?



1



2. Betrouwbaarheid

"Multi-person construction of multi-version software"
- Ploegwerk
- Evolueren of ... uitsterven

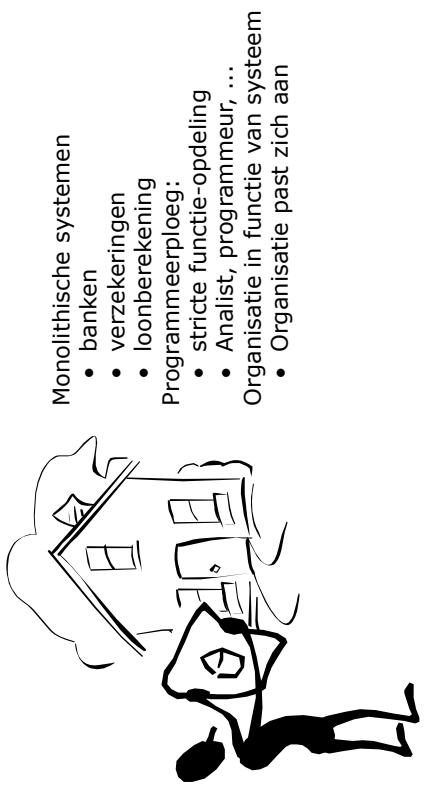
2. Betrouwbaarheid

3

4

"Bouw" een systeem

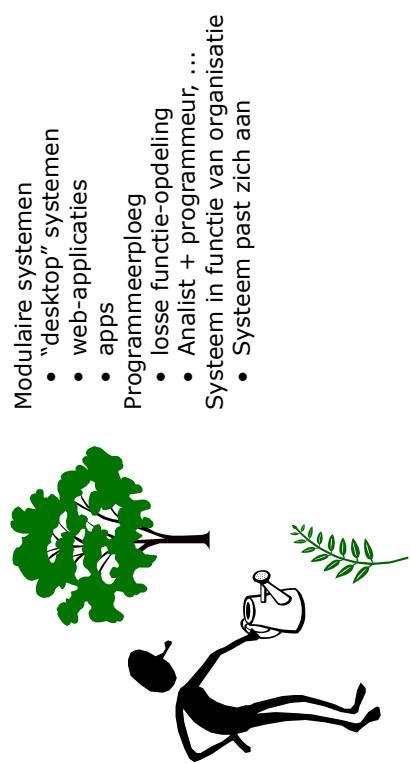
"Ontwikkel" een systeem



2. Betrouwbaarheid

5

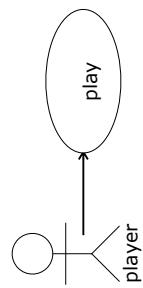
Vereisten



2. Betrouwbaarheid

6

TicTacToe: Specificatie



- 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. Announce winner

2. Betrouwbaarheid

7

8

Vuistregel



Keep It Stupidly Simple (the KISS principle)

Waarom ?

"There are two ways of constructing a software design: one way is to make it so simple that there are obviously no deficiencies, and the other way is make it so complicated that there are no obvious deficiencies."
(C. A. R. Hoare - Turing Award Lecture)

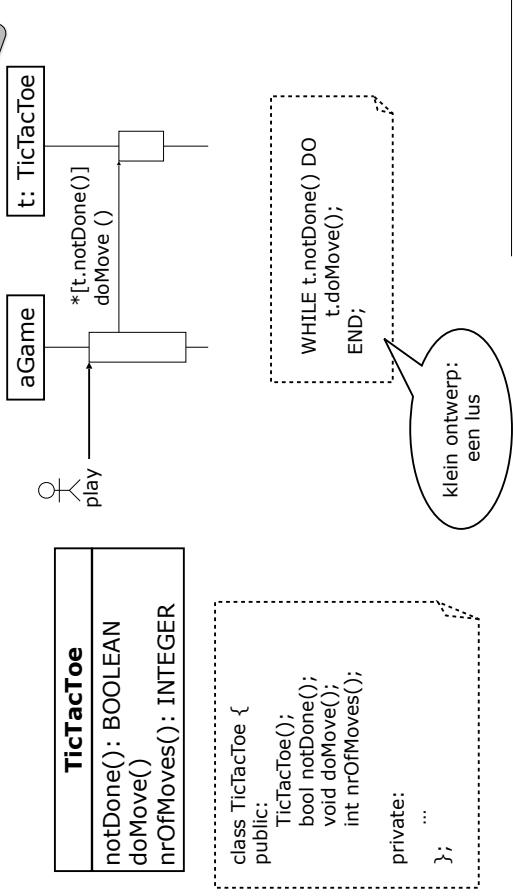
Hoe ?

Begin zo klein mogelijk
⇒ laat ontwerp & implementatie langzaam groeien

2. Betrouwbaarheid

9

TicTacToe10



10

TicTacToe10 in C++

```
class TicTacToe {
public:
    TicTacToe();
    bool notDone();
    void doMove();
    int nrOfMoves();
private:
    int _nrOfMoves;
};

simple implementatie
= een teller
```

```
int TicTacToe::nrOfMoves() {
    return _nrOfMoves;
}
```

Vuistregel



Minstens één test per
"normaal" scenario

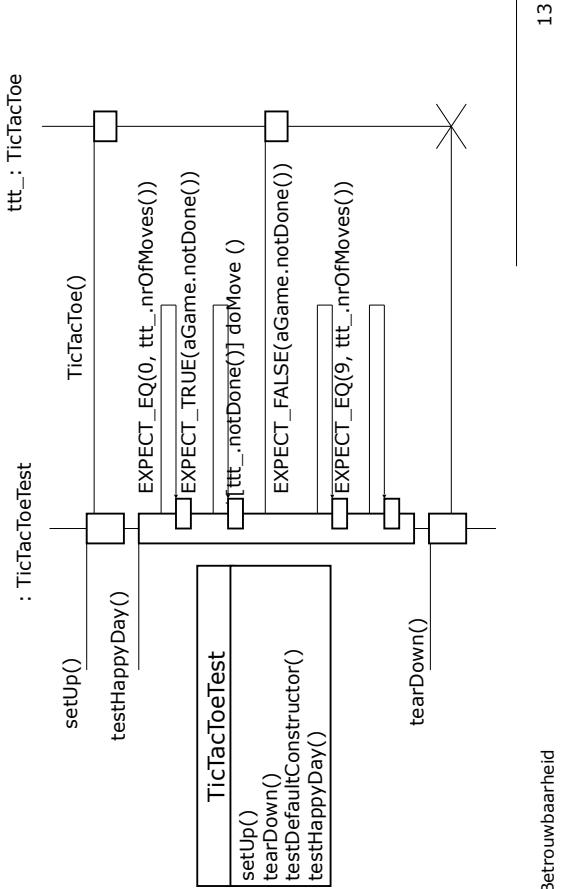
Waaron ?
Minimum veiligheidsnorm
Hoe ?
Controleer de tussentijdse toestand via "EXPECT_..."

11

12

2. Betrouwbaarheid

TicTacToe10: Test (Happy Day)



Test the "happy day" scenario

```
TEST_F(TicTacToeTest, HappyDay) {
    EXPECT_EQ(0, ttt_.nrOfMoves());
    EXPECT_TRUE(ttt_.notDone());
    while (ttt_.notDone()) {
        ttt_.doMove();
    }
    EXPECT_FALSE(ttt_.notDone());
    EXPECT_EQ(9, ttt_.nrOfMoves());
}
```

2. Betrouwbaarheid

Infrastructuur (GoogleTest)

```
class TicTacToeTest: public ::testing::Test {
protected:
    virtual void SetUp() {
    }
    virtual void TearDown() {
    }
    TicTacToe ttt_;
};

... Tests here by invoking macro TEST_F ...

// the main function ... there can only be one :)
int main(int argc, char **argv) {
    ::testing::InitGoogleTest(&argc, argv);
    return RUN_ALL_TESTS();
}
```

13

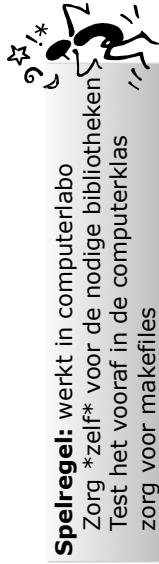
Testresultaat

```
[=====] Running 2 tests from 1 test case.
[-----] Global test environment set-up.
[-----] 2 tests from TicTacToeTest
[ RUN   ] TicTacToeTest.DefaultConstructor
[   OK   ] TicTacToeTest.DefaultConstructor (0 ms)
[ RUN   ] TicTacToeTest.HappyDay
[   OK   ] TicTacToeTest.HappyDay (0 ms)
[-----] 2 tests from TicTacToeTest (0 ms total)

[-----] Global test environment tear-down
[=====] 2 tests from 1 test case ran. (1 ms total)
[  PASSED ] 2 tests.
```

2. Betrouwbaarheid

14



15

16

2. Betrouwbaarheid

Bewust Falende test

```
[ FAILED ] TicTacToeTest.HappyDay (1 ms)
[-----] 2 tests from TicTacToeTest (1 ms total)

[=====] Global test environment tear-down
[=====] 2 tests from 1 test case ran. (1 ms total)
[ PASSED ] 1 test.
[ FAILED ] 1 test, listed below:
[ FAILED ] TicTacToeTest.HappyDay
```



Projectverdediging: Quiz

Tik verander één van de tests
Slaagt de test ? Of faalt de test ?

2. Betrouwbaarheid

Build Rules

target : depends upon
build rule(s)

een make file is een opeenvolging van
“regels”

```
% .o : %.cpp %.h
g++ -I.../gtest/include -c -o $@ $<
```

een.o file is afhankelijk van .h en.cpp file
compilieren met de juiste parameters

Build Rules (2)

• PHONY: non file targets
build rule(s)

.PHONY is een conventie om build targets te
markeren die *geen* bestanden zijn
De rule zal *altijd* afvuren

```
• PHONY : clean
clean :
rm -f *.o TicTacToeMain TicTacToeTests
```

Vaak gebruikte .PHONY targets
“all” “clean” “install”

```
• PHONY : sourceszip
sourceszip :
zip -q -r TicTacToe.zip *.h *.cpp Makefile
```

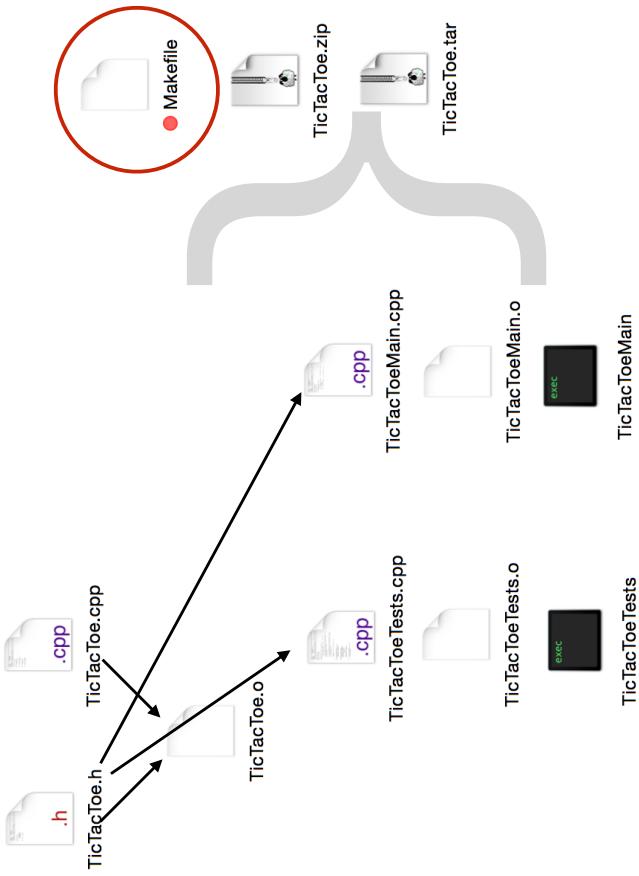
TicTacToeMain : TicTacToe.o TicTacToeMain.o
g++ -o \$@ TicTacToe.o TicTacToeMain.o
De main file is afhankelijk van alle .o files
linken met de juiste parameters

Maak een reservekopie !!!

```
• PHONY : depend
depend :
g++ -MM -I $(INCL) $(SRCS)
```

Genereer de dependencies

17



Build Rules (3)

varname = variable contents
\$ (varname)

We kunnen ook variabelen definieren
... en dan later gebruiken

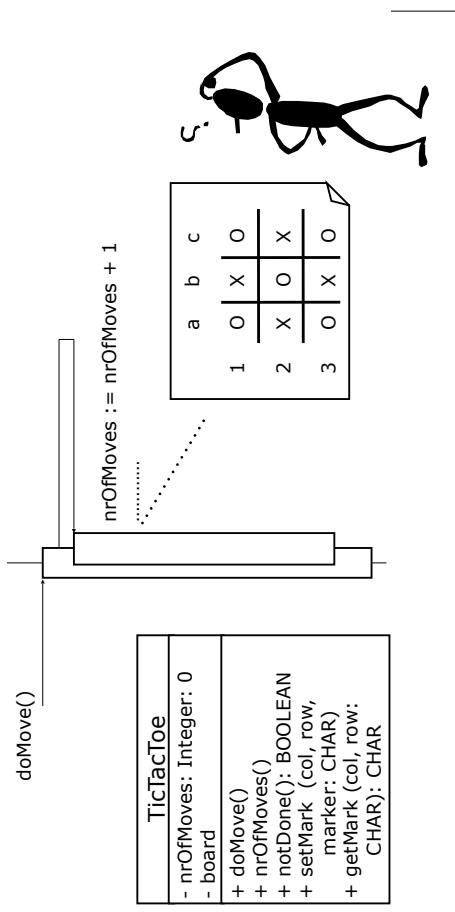
```
CXXFLAGS = -O2 -g -Wall -fmessage-length=0 -I$(INCL)
OBJS = TicTacToe.o
SRCS = TicTacToe.cpp \
        TicTacToeMain.cpp \
        TicTacToeTests.cpp

TicTacToeMain : $(OBJS) TicTacToeMain.o
$(CXX) $(CXXFLAGS) -o $@ $(OBJS) TicTacToeMain.o

.PHONY : echo
echo : @echo CXXFLAGS = $(CXXFLAGS)
@echo CXX = $(CXX)
...
```

2. Betrouwbaarheid

t: TicTacToe



22

Incrementeel Ontwerp

```
void TicTacToe::doMove() {
    char col, row, marker;
    col = (char) (_nrOfMoves % 3) + 'a';
    row = (char) (_nrOfMoves / 3) + '0';
    if (_nrOfMoves % 2) marker = 'X'; else marker = 'O';
    // when _nrOfMoves is odd assign 'X'
    this->setMark(col, row, marker);
    _nrOfMoves++;
}
```

Ontwerp & Implementatie
groeiend langzaam mee !

Vuistregel



21

2. Betrouwbaarheid

TicTacToe11 doMove

We kunnen ook variabelen definieren
... en dan later gebruiken

```
TicTacToeMain : $(OBJS) TicTacToeMain.o
$(CXX) $(CXXFLAGS) -o $@ $(OBJS) TicTacToeMain.o

Een build regel met variabelen
Een build regel met variabelen
echo variabelen
echo variabelen
(debug makefile)
...
```

21

22

Als de functionaliteit
groeit,
dan groeit de test mee

Waarom ? Veiligheidsmaatregel: gecontroleerde groei
Hoe ? Pas bestaande tests aan
⇒ extra tussentijds toestanden controleren

23

24

2. Betrouwbaarheid

Test groeit mee

```
TEST_F(TicTacToeTest, HappyDay) {  
    ...  
    while (ttt_.notDone() ) {  
        ttt_.doMove();  
    }  
    char col, row;  
    bool markIsX = false; // start with 'O'  
    for (col = 'a'; col < 'd'; col++)  
        for (row = '0'; row < '3'; row++) {  
            if (markIsX)  
                EXPECT_EQ('X', ttt_.getMark(col, row));  
            else  
                EXPECT_EQ('O', ttt_.getMark(col, row));  
            markIsX = not markIsX;  
        }  
    ...  
}
```

2. Betrouwbaarheid

Een slechte test ...

```
TEST_F(TicTacToeTest, ManualBADHappyDayTest){  
    cout << "Start of game: ttt.nrofMoves() = "  
    << ttt_.nrofMoves() << endl;  
    cout << "Start of game: ttt.notDone() = "  
    << ttt_.notDone() << endl;  
    while (ttt_.notDone() ) {  
        ttt_.doMove();  
    }  
    char col, row;  
    bool markIsX = false; // start with 'O'  
    for (col = 'a'; col < 'd'; col++)  
        for (row = '0'; row < '3'; row++) {  
            cout << col << " - " << row << ":"  
            << ttt_.getMark(col, row) << " =? ";  
            if (markIsX) cout << "X" << endl; else cout << "O" << endl;  
            markIsX = not markIsX;  
        }  
    cout << "End of game: ttt.nrofMoves() = " << ttt_.nrofMoves() << endl;  
    cout << "End of game: ttt.notDone() = " << ttt_.notDone() << endl;  
}
```

2. Betrouwbaarheid



Manuele Verificatie van Tests

```
Start of game: ttt.nrofMoves() = 0  
Start of game: ttt.notDone() = 1  
a - 0: O ==? O  
a - 1: X ==? X  
a - 2: O ==? O  
a - 3: X ==? X  
Schrijf *noot* testcode  
die manuele verificatie  
van de uitvoer vereist  
C
```

Hoe weet ik of
deze uitvoer
correct is ?

```
Start of game: ttt.nrofMoves() = 0  
Start of game: ttt.notDone() = 1  
a - 0: O ==? O  
a - 1: X ==? X  
a - 2: O ==? O  
a - 3: X ==? X  
Schrijf *noot* testcode  
die manuele verificatie  
van de uitvoer vereist  
C
```

Hoe weet ik of
deze uitvoer
correct is ?

Vuistregel



Een test produceert zo
weinig mogelijk uitvoer
"PASSED"
"FAILED"

```
...  
    Waarom ?  
    Hoe dan wel ?  
    Tests antwoorden  
    • "PASSED" = alle tests zijn goed verlopen (welke ?)  
    • "FAILED" = minstens één test heeft gefaald (welke ?)
```

2. Betrouwbaarheid

```
... ook TestDefaultConstructor
```

2. Betrouwbaarheid

26

26

27

Evaluatiecriteria

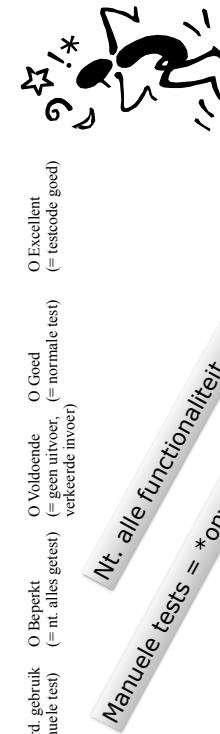
TicTacToe11: Klasse Interface

Inleiding Software Engineering (20?? - 20??)

Student1 : student1 - Student2: student2 - Student3: student3

Commentaar:

Test



Manuele tests = *onvoldoende*

2. Betrouwbaarheid

29

Een klein foutje ...

TicTacToe.h

```
//The state of the game is represented as 3x3 array  
//of chars marked ' ', 'X', or 'O'.  
//We index the state using chess notation, i.e.,  
//column is 'a' through 'c' and row is '1' through '3'.*)  
void setMark(char col, char row, char marker);  
char getMark(char col, char row);
```

2. Betrouwbaarheid

30

Een klein foutje ... grote gevolgen (1/2)



WIKIPEDIA
The Free Encyclopedia

Mars Climate Orbiter

From Wikipedia, the free encyclopedia

The **Mars Climate Orbiter** (formerly the **Mars Surveyor '98 Orbiter**) was a 338 kilogram (750 lb) robotic space probe launched by NASA on December 11, 1998 to study the Martian climate, atmosphere, surface changes and to act as the communications relay in the Mars Surveyor '98 program, for Mars Polar Lander. However, on September 23, 1999, communication with the spacecraft was lost as the spacecraft went into orbital insertion, due to a navigational error. The spacecraft encountered Mars at an improperly low altitude, causing it to incorrectly enter the upper atmosphere and disintegrate.^{[1][2]}

[Contents](#) [hide]
[1 Mission background](#)

Zijn alle waardes van type CHAR legale col & row ?
⇒ NEE: "a" .. "c" of "1" .. "3"
Is elke waarde van type CHAR een legale marker ?
⇒ NEE: " " , "X" of "O"
Wat is het verband tussen "getMark" en "setMark" ?
⇒ resultaat "getMark" is hetgeen vroeger gezet werd via "setMark"

```
TicTacToe  
- nrOfMoves: Integer: 0  
- board  
+ doMove()  
+ nrOfMoves()  
+ notDone(): BOOLEAN  
+ setMark (col, row, marker: CHAR)  
+ getMark (col, row: CHAR): CHAR
```



Log in / create account

Q

Read Edit View history Search

[Article](#) [Talk](#)

[Read](#)

[Edit](#)

[View history](#)

[Search](#)

Log in / create account

Q

Read Edit View history Search

[Article](#) [Talk](#)

[Read](#)

[Edit](#)

[View history](#)

[Search](#)

Tweede Evaluatie

Tweede Evaluatie

Inleiding Software Engineering (20?? - 20??)

Student1 : student1 - Student2: student2 - Student3: student3

Commentaar:

Test



Manuele tests = *onvoldoende*

2. Betrouwbaarheid

31

2. Betrouwbaarheid

31

32

Een klein foutje ... grote gevolgen (2/2)

[bron: History's Most (In)Famous Software Failures]

1988 – Buffer overflow in Berkeley Unix finger daemon.
he first internet worm (the so-called Morris Worm) infects between 2,000 and 6,000 computers in less than a day by taking advantage of a buffer overflow. The specific code is a function in the standard input/output library routine called gets() designed to get a line of text over the network. Unfortunately, gets() has no provision to limit its input, and an overly large input allows the worm to take over any machine to which it can connect. Programmers respond by attempting to stamp out the gets() function in working code, but they refuse to remove it from the C programming language's standard input/output library, where it remains to this day.

1998 – E-mail buffer overflow
Several E-mail systems suffer from a "buffer overflow error", when extremely long e-mail addresses are received. The internal buffers receiving the addresses do not check for length and allow their buffers to overflow causing the applications to crash. Hostile hackers use this fault to trick the computer into running a malicious program in its place.

2. Betrouwbaarheid

33



- Controleer de waarden van de argumenten die je binnenkrijgt (PRE)
- Controleer de waarde van het resultaat dat je teruggeeft (POST)

Vuistregel

Waaron ? Meeste fouten door interpretatieproblemen van de interface

Hoe ? Schrijf pre- en postcondities bij elke procedure
⇒ PRE: REQUIRE(<boolse expressie>, <boodschap>
⇒ POST: ENSURE(<boolse expressie>, <boodschap>)

2. Betrouwbaarheid

34

Design By Contract

```
#include "DesignByContract.h"
REQUIRE en ENSURE
...
char TicTactoe::getMark(char col, char row) {
    ...
    REQUIRE(( 'a' <= col) && (col <= 'c'), "col must be between 'a' ...
        ...
        REQUIRE(( '1' <= row) && (row <= '3'), "row must be between '1' ...
            ...
            REQUIRE(( '1' <= col) && (col <= 'c'),
                ...
                REQUIRE(( '1' <= row) && (row <= '3'),
                    ...
                    REQUIRE(( '1' <= col - 'a') && (int) row - '1' <= col,
                        ...
                        ENSURE(( 'x' == result) || ('o' == result) || (' ' == result),
                            ...
                            "getMark returns 'x', 'o' or ' '");
                    ...
                    return result;
                }
            }
        }
    }
}
```

Contracten = Interface

Bestand: TicTactoe.h

```
char getMark(char col, char row);
    ...
    // REQUIRE(( 'a' <= col) && (col <= 'c'), "col must be between 'a' ...
    // REQUIRE(( '1' <= row) && (row <= '3'), "row must be between '1' ...
    // ENSURE(( 'x' == result) || ('o' == result) || (' ' == result) || (' ' == result),
    //         ...
    //         "getMark returns 'x', 'o' or ' '");
```

- naamconventie: gebruik "result" in post conditie



KOPIEER CONTRACTEN IN .h file

- deel van de klasse-interface
⇒ CONTRACT met de buitenwereld
- (soms automatisch gegenereerd
⇒ steeds synchroon met de code)

2. Betrouwbaarheid

35

36

Vuistregel



Waarom ?
Hoe ?

contract verifieerbaar door externe partijen (modules, ...)
Private (nt. geëxporteerde) declaraties extern niet toegankelijk
⇒ Denk goed na over wat je pre- & postcondities nodig hebben

2. Betrouwbaarheid

Private declaraties

```
void TicTacToe::setMark(char col, char row, char marker) {  
    REQUIRE( ('a' <= col) && (col <= 'c'), "col must be between 'a' and 'c'");  
    REQUIRE( ('1' <= row) && (row <= '3'), "row must be between '1' and '3'");  
    REQUIRE( ('X' == marker) || ('O' == marker) || (' ' == marker),  
            "marker must be 'X', 'O' or ' '");  
    _board[ (int) col - 'a' ][(int) row - '1' ] = marker;  
    ENSURE( _board[ (int) col - 'a' ][(int) row - '1' ] == marker,  
           "setMark postcondition failure");  
}
```



board is niet geëxporteerd
⇒ contract niet verifieerbaar

37

38

Vuistregel



Volgorde van oproepen

```
void TicTacToe::setMark(char col, char row, char marker) {  
    REQUIRE( ('a' <= col) && (col <= 'c'), "col must be between 'a' and 'c'");  
    REQUIRE( ('1' <= row) && (row <= '3'), "row must be between '1' and '3'");  
    REQUIRE( ('X' == marker) || ('O' == marker) || (' ' == marker),  
            "marker must be 'X', 'O' or ' '");  
    _board[ (int) col - 'a' ][(int) row - '1' ] = marker;  
    ENSURE( (getMark( col, row ) == marker),  
           "setMark postcondition failure");  
}
```

Postconditie van setMark gebruikt getMark
⇒ setMark oproepen voor getMark

39

40

Waarom ?
Belangrijk, maar niet te lezen in een "platte" klasse-interface
Hoe ?
post-conditie van de voorgaande ⇒ pre-conditie van de volgende

2. Betrouwbaarheid

Vuistregel



Gebruik pre-conditie om de initialisatie van objecten te controleren

Waarom?
initialisatie vergeten: veel voorkomende & moeilijk te vinden fout
Hoe?
pre-conditie met "properlyInitialized"? Controleer een self-pointer

2. Betrouwbaarheid

41

"properlyInitialized" via self pointer

```
class TicTacToe {  
    ...  
    private:  
        TicTacToe * _initCheck;  
        ...  
        TicTacToe::TicTacToe() {  
            _initCheck = this;  
            ...  
        }  
        bool TicTacToe::properlyInitialized() {  
            return _initCheck == this;  
        }  
    }  
}
```

Object heeft een extra pointer ...

```
        ...  
        Wijst normaal naar zichzelf  
        na initialisatie  
        ...  
        En kan achteraf  
        geverifieerd worden
```

42

properlyInitialized pre- and postcondition

```
class TicTacToe {  
public:  
    TicTacToe();  
    // ENSURE(properlyInitialized(), "constructor must end ...");  
    "properlyInitialized"  
    ⇒ postconditie alle constructors  
    ⇒ verifieer in testDefaultConstructor  
  
    bool notDone();  
    // REQUIRE(this->properlyInitialized(),  
    // "TicTacToe wasn't initialized when calling notDone");  
    "properlyInitialized"  
    ⇒ precondition andere procedures  
    void doMove();  
    // REQUIRE(this->properlyInitialized(),  
    // "TicTacToe wasn't initialized when calling doMove");  
    ...  
}
```

Zie TicTacToe.h

```
TicTacToe  
properlyInitialized () : BOOLEAN  
notDone()  
doMove()  
nrOfMoves()  
getMark (col, row: CHAR): CHAR  
setMark (col, row, marker: CHAR)
```

<<precondition>>
properlyInitialized() &
("a" <= col) & (col <= "c") &
("1" <= row) & (row <= "3")
<<postcondition>>
(result = "O") OR (result = "X")
OR (result = "")
<<postcondition>>
getMark(col, row) = marker

<<precondition>>
properlyInitialized() &
("a" <= col) & (col <= "c") &
("1" <= row) & (row <= "3")
& ((marker = "O") OR (marker = "X"))
OR (marker = "")
<<postcondition>>
getMark(col, row) = marker

43

2. Betrouwbaarheid

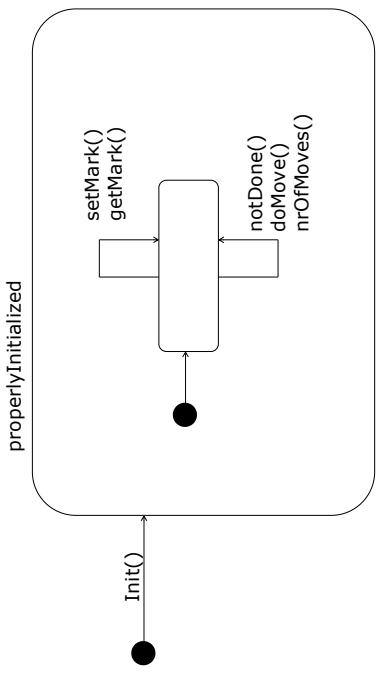
TicTacToe12: Contracten

44

2. Betrouwbaarheid

Interpretatie Interface

*Notatie = UML
(Zie boek "Using UML ...")*



Dit is een eindige automaat (cfr. "Talen en Automaten")

2. Betrouwbaarheid

Vuistregel



**schrijf ook een test voor
ContractViolations**

Waarom ?
kijk na of de ABNORMAL END wel degelijk gebeurt
Hoe ?
schrijf tests met oproep illegale waardes
gebruik EXPECT_DEATH ter verificatie
`EXPECT_DEATH(ttt_.getMark('a' - 1, '1'), "failed assertion");`

Evaluatiecriteria

Interleiding Software Engineering (20?? - 20??)

Student1 - Student2 - Student3 - Student4 - Student5

Commentaar:

Contracten

O Nt aanwezig (= geen assert)

O Vrkld. gebruik (= alleen pre-post)

O Beperkt (= alleen pre)

O Voidende (= soms pre & post) (= vaak pre & post)

O Goed (= alleen pre)

O Excellent (= soms pre & post) (= vaak pre & post)

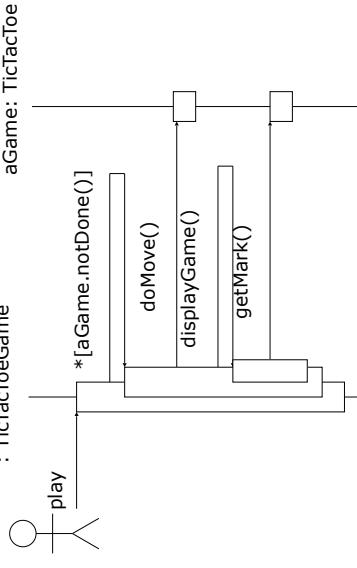
Commentaar:



Gebruik van private/protected = *onvoldoende*
= Niet zichtbaar in .h file
Verkeerd gebruik = *onvoldoende*

46

TicTacToe12



2. Betrouwbaarheid

47

48

2. Betrouwbaarheid

Vuistregels (Samenvatting)

ONTWERP

- Keep It Stupidly Simple (the KISS principle)



TESTEN

- Minstens één test per "normaal" scenario
- Als de functionaliteit groeit, dan groeit de test mee
- Een test produceert zo weinig mogelijk uitvoer

CONTRACTEN

- Controleer de waardes van de argumenten die je binnenkrijgt
- Controleer de waarde van het resultaat dat je teruggeeft
- Kopieer contracten in .h file
- Contracten gebruiken alleen publieke (geëxporteerde) declaraties
- Contracten bepalen de volgorde van oproepen.
- Gebruik pre-conditie om de initialisatie van objecten te controleren
 - schrijf ook een test voor ContractViolations (commentaar)

Complexe Interacties

Een optimale werkverdeling



3. Aanpasbaarheid

3. Aanpasbaarheid

versie 13 (displayGame)

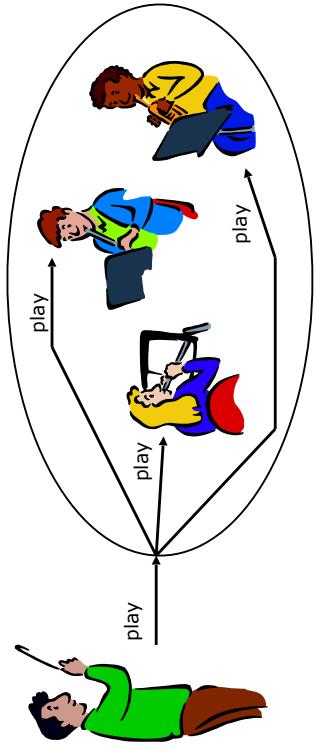
- 0, 1, infinity principle
 - say things once and only once
- versie 14 (displayGame)
- coupling / cohesion
 - [testen] ASCII uitvoer
- versie 15 (Player)
- domeinmodel

versie 16 (Player.moves)

- vermijd gebruikersinvloer
- versie 17 (Player.winner())
- Basisfunctionaliteit

Conclusie

3. Aanpasbaarheid



3. Aanpasbaarheid

Vereisten



3

2.Betrouwbaarheid



Vereisten	Technieken
• Betrouwbaarheid	• Testen + Contracten
• Aanpasbaarheid	• Objectgericht ontwerp

Werktuigen	
• "Build"	• Make
• Editor	• Eclipse
• Debugger	
• Documentatie	• Doxygen

4

Magic Numbers

```
void displayGame(TicTacToe& ttt) {  
    ...  
    for (row = '1'; row <= '3'; row++) {  
        cout << row;  
        for (col = 'a'; col <= 'c'; col++) {  
            ...  
        }  
    }  
  
    void TicTacToe::doMove() {  
        ...  
        col = (char) (_nrOfMoves % 3) + 'a';  
        row = (char) (_nrOfMoves / 3) + '1';  
        ...  
    }  
  
TEST_F(TicTacToeTest, HappyDay) {  
    ...  
    for (col = 'a'; col <= 'c'; col++)  
        for (row = '1'; row <= '3'; row++) {  
            ...  
        }  
}
```

3. Aanpasbaarheid

5



Vuistregel

"0, 1, infinity" principle
Allow - none of foo,
- one of foo, or
- any number of foo.

6

Waarom ?

zet geen arbitraire limieten

⇒ Vroeg of laat zullen deze limieten aangepast moeten worden

Refactor:

creëer een constante die de arbitraire waarde voorstelt



Vuistregel

"Keep It Variant op
Stupidly Simple"

say things
once and only once

Waarom ?

duplicatie en redundantie

⇒ veranderen moet op twee, drie ... plaatsen
(een kleine wordt vroeg of laat vergeten)

Refactor:
creëer abstractie via naam, functie, klasse, ...

Y2K

kalenderjaar = laatste
twee cijfers

'98 '99 ?? '00 '01

3. Aanpasbaarheid

7

Y2K Cost in Perspective
(Sources: Gartner Group and Congressional Research Service)

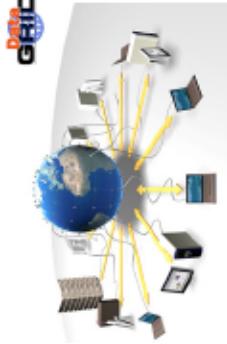
- (World War II: \$4200 billion
- Millennium bug (Y2K): \$600 billion
- Vietnam conflict: \$500 billion
- Kobe earthquake: \$100 billion
- Los Angeles earthquake: \$60 billion

8

3. Aanpasbaarheid

IPv4 address space

- Internet Protocol (version 4)
 - address ruimte = 2^{32}
 - $\approx 4 \times 10^9 \approx 4$ miljard
 - 3 februari 2011
 - laatste nummer toegewezen door Internet Assigned Numbers Authority (IANA)



Magic Numbers \Rightarrow Constantes

```
Internet Protocol (version 6)
address ruimte =  $2^{128}$ 
≈  $3,4 \times 10^{38}$ 
✓ aardbewoner
± 50 quadriljard adressen

void displayGame(TicTacToe& ttt) {
    ...
    for (row = minRow; row <= maxRow; row++) {
        cout << "row";
        for (col = minCol; col <= maxCol; col++) { ...
    }
```

```
void TicTacToe::doMove() {
    ...
    col = (char) (_nrOfMoves % 3) + minCol;
    row = (char) (_nrOfMoves / 3) + minRow;
    ...
}
```

```
TEST_F(TicTacToeTest, HappyDay) {
    ...
    for (col = minCol; col <= maxCol; col++) {
        for (row = minRow; row <= maxRow; row++) {
            ...
        }
    }
}
```

9

3. Aanpasbaarheid

10

Pass by Reference vs. Pass by Value (1/2)

```
TicTacToe13
void displayGame(TicTacToe& ttt) {
    ...
}
```

TicTacToe.cpp:44: failed assertion
TicTacToe wasn't initialized ...
TicTacToe::numberOfMoves = Abort trap: 6



```
TicTacToe::numberOfMoves = 1
a   b   c
-----
```

1	O	
2		
3		

```
TicTacToe::numberOfMoves = 2
a   b   c
-----
```

1	O	X
2		
3		

Pass by Reference vs. Pass by Value (2/2)

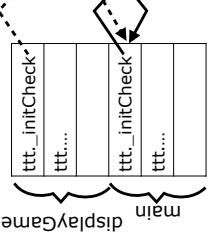
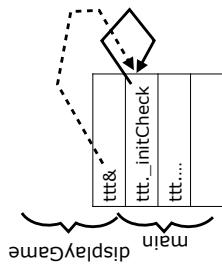
```
properlyInitialized ⇒ defensieve programmeerstijl
void displayGame(TicTacToe& ttt) {
    ...
}

int main(...) {
    TicTacToe ttt;
    ...
}
```

```
void displayGame(TicTacToe& ttt) {
    ...
}

int main(...) {
    TicTacToe ttt;
    ...
}
```

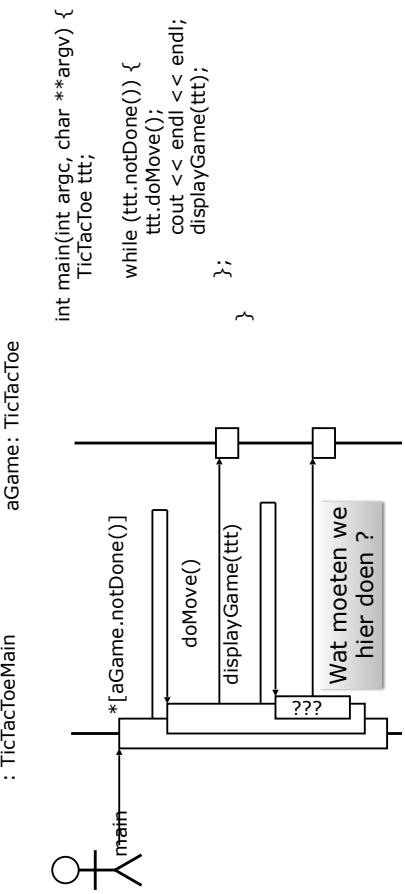
Pass by Reference



3. Aanpasbaarheid

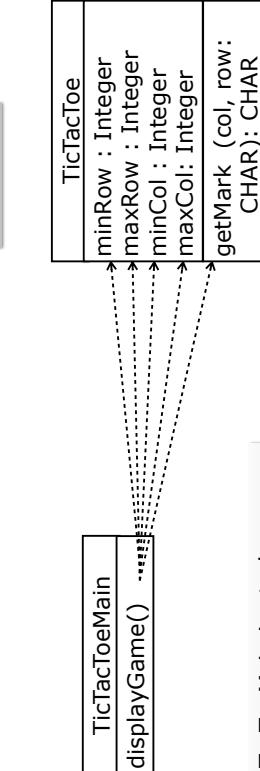
11

TicTacToe13



3. Aanpasbaarheid

Sterke Koppeling



TicTacToeMain is sterk gekoppeld aan TicTacToe
• Er zijn veel afhankelijkheden van TicTacToeMain naar TicTacToe
• Een verandering aan TicTacToe betekent meestal dat we ook TicTacToeMain moeten aanpassen

3. Aanpasbaarheid

Sterke Koppeling (code)

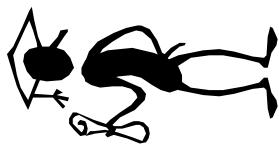
```

void displayGame(TicTacToe& ttt) {
    char col, row;
    cout << "TicTacToe numberofMoves = " << ttt.nrofMoves() << endl;
    cout << "   a   b   c   " << endl;
    cout << "   -----   " << endl;
    for (row = minRow; row <= maxRow; row++) {
        cout << row;
        for (col = minCol; col <= maxCol; col++) {
            cout << " | " << ttt.getMark(col, row);
        }
        cout << " | " << endl;
    }
    cout << "-----" << endl;
}
  
```

Afhandelbaar van 4 constantes & 1 operatie

14

Vuistregel



Waarom ?

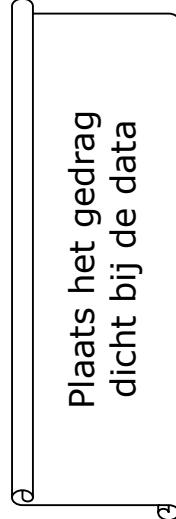
- veranderingen zullen eerder lokaal effect hebben
⇒ zwakke koppeling

Refactor:

- Routines die veel "get" operaties oproepen
⇒ verplaatst naar de corresponderende klasse.

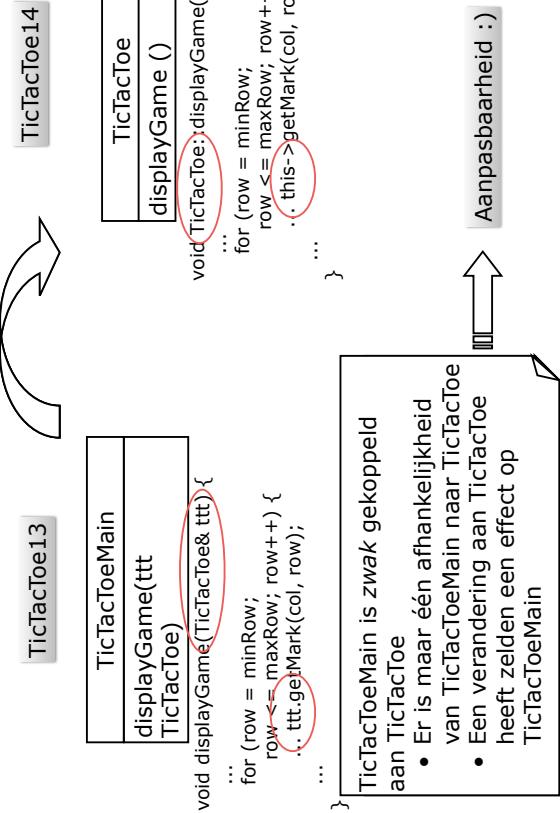
15

3. Aanpasbaarheid



16

Zwakke Koppeling



3. Aanpasbaarheid

Vuistregel



Nooit user-interface code
in de basisklassen

Waarom ?

- Minder kans op veranderingen in basisklassen
(user-interface wordt vaak veranderd)
- Refactor:
 - Extra parameter als in/uitvoerkanaal naar de buitenwereld

Koppeling via cout

```
void TicTacToe::displayGame() {  
    char col, row;  
    REQUIRE(this->properlyInitialized(),  
           "TicTacToe wasn't initialized when calling displayGame");  
    std::cout << "TicTacToe numberofMoves = " << this->nrOfMoves();  
    std::cout << endl;  
    std::cout << "-----" << endl;  
    for (row = minRow; row <= maxRow; row++) {  
        std::cout << row;  
        for (col = minCol; col <= maxCol; col++) {  
            std::cout << " | " << this->getMark(col, row);  
        }  
        std::cout << endl;  
    }  
    std::cout << " ----- " << endl;  
}
```

TicTacToe is afhankelijk van cout
⇒ alleen uitvoer op console 😞

18

Koppeling via ostream

```
void TicTacToe::writeOn(std::ostream& onStream) {  
    char col, row;  
    REQUIRE(this->properlyInitialized(),  
           "TicTacToe wasn't initialized when calling displayGame");  
    onStream << "TicTacToe numberofMoves = " << this->nrOfMoves();  
    std::endl;  
    onStream << "-----" << endl;  
    for (row = minRow; row <= maxRow; row++) {  
        onStream << row;  
        for (col = minCol; col <= maxCol; col++) {  
            onStream << " | " << this->getMark(col, row);  
        }  
        onStream << endl;  
    }  
    onStream << " ----- " << endl;
```

Extra parameter als uitvoerkanaal
+ betere naamgeving

20

3. Aanpasbaarheid

19

3. Aanpasbaarheid

Cohesie

TicTacToe14

```
<<precondition>>
propertyInitialized() &
("a" <= col) & (col <= "c") &
("1" <= row) &
(row <= "3")
```

```
<<postcondition>>
(result = "O") OR (result =
 "X") OR (result = " ")
```

```
<<precondition>>
...
<<postcondition>>
getMark (col, row) =
marker
```

TicTacToe();
getMark (col, row: CHAR): CHAR
setMark (col, row, marker: CHAR): CHAR
notDone (): BOOLEAN
doMove ()
writeOn (o: ostream)

TicTacToe is redelijk cohesief
• 1 operatie gebruiken implieert ook het gebruik van alle andere
• alle operaties zijn nodig/nuttig
⇒ veranderingen aan de interface zijn weinig waarschijnlijk

3. Aanpasbaarheid

21

Testen van ASCII uitvoer (TicTacToe14)

```
TicTacToe();
getMark (col, row: CHAR): CHAR
setMark (col, row, marker: CHAR): CHAR
notDone (): BOOLEAN
doMove ()
writeOn (o: ostream)
```

TicTacToe

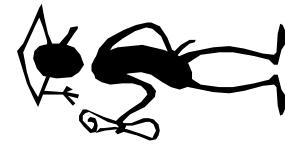
Hoe testen we de uitvoer ?

Als de functionaliteit groeit,
dan groeit de test mee

- Waaron ?
• Veel & frequent testen ⇒ makkelijk om steeds uitvoer te testen
Hoe ?
• Verwachte uitvoerdata wordt één maal manueel gecreëerd
• vergelijk verwachte uitvoer met gegenereerde uitvoer

3. Aanpasbaarheid

23



Vuistregel

3. Aanpasbaarheid

22

?? Ideaal ??
= een component die niks doet
⇒ perfectie is niet haalbaar

22

te MAXIMALISEREN
⇒ veranderingen zijn minder
waarschijnlijk

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

24

Testen van uitvoer

Testing of FileCompare

```
TEST_F(TicTacToeTest, OutputHappyDay) {
    ASSERT_TRUE(directoryExists("testoutput"));
    //if directory doesn't exist then no need in proceeding with the test
    ofstream myfile;
    myfile.open("testoutput/happyDayOutput.txt");
    while (ttt_.notDone()) {
        ttt_.doMove();
        ttt_.writeOn(myfile);
    }
    myfile.close();
    EXPECT_TRUE(
        FileCompare("testoutput/happyDayExpectedOut.txt",
                    "testOutput/happyDayOutput.txt"));
}
```

Schrijf alles op een bestand ...

... en vergelijk met wat je verwacht

code voor "DirectoryExists" en "FileCompare" → Zie TicTacToe14

3. Aanpasbaarheid

25

3. Aanpasbaarheid

26



Test the helpfunctions in de tests !

TicTacToe15

Use Case 1: play

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



Waar voegen
we dit bij ?

- 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. Announce winner



Vuistregel

Maak een model van
het probleem domein
= het DOMEINMODEL

Tips:

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

3. Aanpasbaarheid

27

3. Aanpasbaarheid

28

Domeinmodel

```

stateDiagram-v2
    [*] --> play
    play -- "play" --> choice
    choice -- "player" --> play
    choice -- "player" --> choice
    choice -- "play" --> play
  
```

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

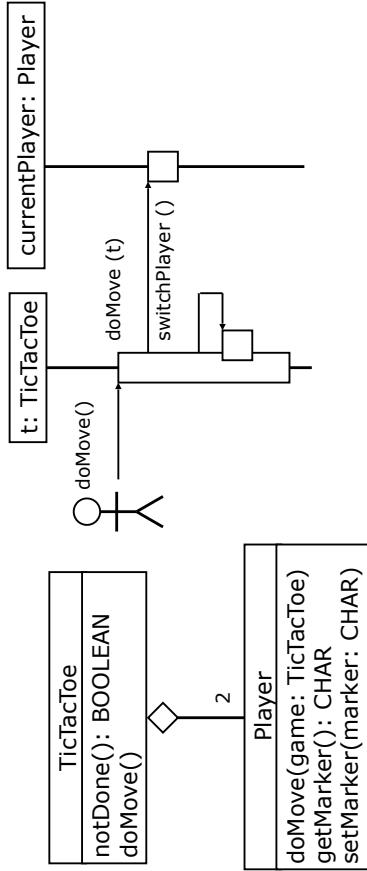
1. Two players start up a game (First is "O"; other is "X")
2. WHILE game not done
 - 2.1 Current player makes move
 - 2.2 Switch current player
3. Announce winner

Legende

- Substantief
- Werkwoord

TTT15: Player

- 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. *Announce* winner



B. Aanpasbaarheid

29

3. Anpasbaarheid

30

Forward Declaration

```
class TicTacToe; // forward declaration
class TicTacToePlayer {
    ...
    void doMove(TicTacToe& game);
};

private:
    TicTacToePlayer * _initCheck; // use pc
    char _marker;
};

class TicTacToe {
    ...
    private:
        ...
        TicTacToePlayer _players [2];
};
```



Opgelet: Magic Number

Verplaats van TicTacToe naar Player Tests blijven open 😊

Tests blijven lopen ...

TicTacToe & TicTactoePlayer
circulair afhankelijk !
=> forward declaration

```
void TicTacToePlayer::doMove(TicTacToe& game) {
    REQUIRE(this->properlyInitialized(),
            "TicTacToePlayer wasn't initialized when calling getMarker");
    REQUIRE((game.properlyInitialized(),
            "game wasn't initialized when passed to Player->doMove");

    char col, row;
    col = (char)(game.nrOfMoves() % 3) + minCol;
    row = (char)(game.nrOfMoves() % 3) + minRow;
    game.setMark(col, row, _marker);
}
```

Verplaats van TicTacToe naar Player Tests blijven lopen

30

3. Anpasbaarheid

31

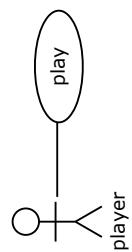
3. Anpasbaarheid

32

TicTacToe16

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. Announce winner



Hoe verkrijgen we
gebruikersinvoer ?



Een test verwacht géén gebruikersinvoer

3. Aanpasbaarheid

Waarom ?

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

Hoe dan wel ?

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

33

34

TTT17: Reeks van zetten

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

een reeks zetten
=>invoerdata door (test)programma

```
int main(int argc, char **argv) {
    TicTacToe ttt;
    ttt.setMoves("a1c1b2a3c3", "b1a2c2b3");
    while (ttt.notDone()) {
        ttt.doMove();
        cout << endl << endl;
        ttt.writeOn(cout);
    }
}
```

```
// Tests the "happy day" scenario
TEST_F(TicTacToeTest, HappyDay) {
    EXPECT_EQ(0, ttt_.nrofMoves());
    EXPECT_TRUE(ttt_.notDone());
    ttt_.setMoves("a1c1b2a3c3", "b1a2c2b3");
    while (ttt_.notDone()) {
        ttt_.doMove();
        char col, row;
        bool markIsX = false; // start with 'O'
        for (col = minCol; col <= maxCol; col++)
            for (row = minRow; row <= maxRow; row++) {
                if (markIsX)
                    EXPECT_EQ('X', ttt_.getMark(col, row));
                ...
            }
    }
}
```

Vuistregel

Use Case 1: play

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

Een test verwacht géén gebruikersinvoer

3. Aanpasbaarheid

35

3. Aanpasbaarheid

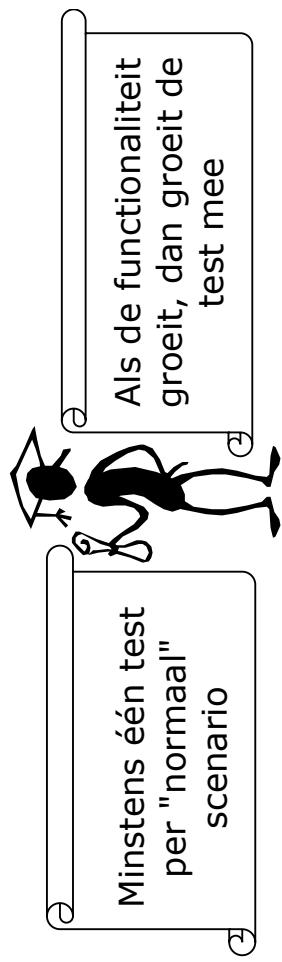
36

Vuistregels (vorige week)

Tests groeien mee ...

```
// Tests whether a given string of moves is legal
TEST_F(TicTacToeTest, LegalMoves) {
    EXPECT_TRUE(tttPlayer_.properlyInitialized());
    EXPECT_TRUE(legalMoves("") );
    EXPECT_TRUE(legalMoves("a1" ) );
    ...
    EXPECT_TRUE(legalMoves("a1a2a3b1b2b3c1c2c3" ) );
    ...
    // illegal moves
    EXPECT_FALSE(legalMoves(" " ) );
    ...
    EXPECT_FALSE(legalMoves("b1c4a1" ) );
}

// Tests the "happy day" scenario for a player
TEST_F(TicTacToeTest, HappyDayPlayer) {
    EXPECT_TRUE(tttPlayer_.properlyInitialized());
    tttPlayer_.setMarker('O' );
    ...
    EXPECT_EQ('O', tttPlayer_.getMarker() );
    ...
}
```



Waarom ? Minimum veiligheidssnorm
Hoe ? Controleer de tussentijdse toestand via "EXPECT_..."

3. Aanpasbaarheid

37

38

TicTacToe17

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. Announce winner

Announce winner
Waar voegen we dit bij ?

Conclusie

Erst 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 probleem domein

goed ontwerp

39

3. Aanpasbaarheid

40

Vuistregels

- Ontwerpen
 - “0, 1, infinity” principle
 - Say things once and only once
 - Plaats het gedrag dicht bij de data
 - Nooit user-interface code in de basisklassen
 - *Zeker* niet voor invoer — frequent testen
 - Maak een model van het probleem domein (= het domeinmodel)
 - zelfstandige naamwoorden & werkwoorden als indicatoren

Testen

- Test lange uitvoer door het vergelijken van files
 - Test the hulpfuncties in de tests !
 - Een test verwacht géén gebruikersinvoer



Evaluatie Criteria (Tests)

Inleiding Software Engineering (20?? - 20??)

Student 1: student 1 - Student 2: student 2 - Student 3: student 3

Commentaar:

- Tests**
 - Nr aanwezig (= geen tests)
 - O Vlkd, gebruik (= manuele test)
 - O Beperkt (= n.t. alles getest)
 - O Volledige (= geen uitvoer, verkeerd, invoer)
 - O Goed (= normale test)
 - O Excellent (= testcode goed)

Een test verwacht géén gebruikersinvoer
Test lange uitvoer door het vergelijken van files

3. Aanpasbaarheid

41

3. Aanpasbaarheid

42

Evaluatie Criteria (Ontwerp)

- Objectgericht Ontwerp** [Alles slankkruisen!]
 - O Geen
 - O Inheritance
 - O Reuse
 - O Lijsten
 - O Goed ADT
 - O Obj, collaboratie
 - O Controle Obj.
 - printobjecten, iterators

Maak een model van het probleem domein

Nooit user-interface code in de basisklassen

Plaats het gedrag dicht bij de data

"0, 1, infinity" principle

Say things once and only once

- Data encapsulatie**
 - O geen
 - O aparte files
 - O rechtstreekse access
 - O getters/setters

Maak een model van het probleem domein

Nooit user-interface code in de basisklassen

Plaats het gedrag dicht bij de data

"0, 1, infinity" principle

Say things once and only once

- Smelly code**
 - O Long method
 - O Long parameter list
 - O Inappropriate intimacy

Maak een model van het probleem domein

Nooit user-interface code in de basisklassen

Plaats het gedrag dicht bij de data

"0, 1, infinity" principle

Say things once and only once

- Gefundeerde beslissingen**
 - O Geen reden
 - O intuïtie
 - O doordacht

Maak een model van het probleem domein

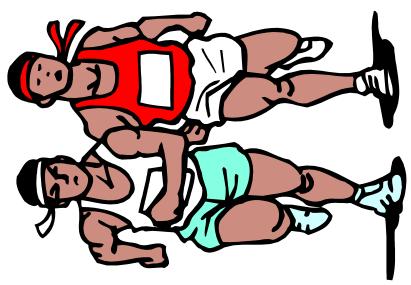
Nooit user-interface code in de basisklassen

Plaats het gedrag dicht bij de data

"0, 1, infinity" principle

Say things once and only once

Hoe snel loopt iemand de 100 meter ?



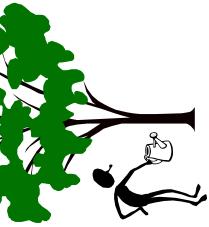
4. Planning

4. Planning

- Tijdsschatting
 - + Analogie & Decompositie
 - + Empirische schatting
 - Plan 2.0 & Plan 2.1
- Conclusie
- TicTacToe17 en TicTacToe18
 - Player. winner()
- Enkele vuistregels
 - + Hollywood principe
 - + 3-lagen architectuur — TicTacToe19 & TicTacToe20
 - ⇒ invoer tests + uitvoer tests + domein test
 - + Hollywood principe - revisited (TicTacToe21)
 - Template Method and Hook Method
 - + Contracten & Interface
- Eclipse
- Doxygen

1

Vereisten

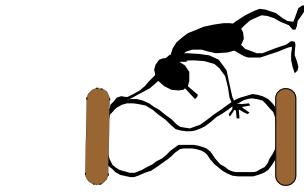


Schatting door analogie

- Analogie
- Schatting = tijd gelijkaardig project
 - Wanneer gelijkaardig ?
 - + Zelfde problemdomein
 - + Zelfde mensen
 - + Zelfde technologie

Vereisten	Technieken
<ul style="list-style-type: none">• Betrouwbaarheid• Aanpasbaarheid• Planning	<ul style="list-style-type: none">• Testen + Contracten• Objectgericht ontwerp• Tijdsschatting
Werk具gen	
<ul style="list-style-type: none">• "Build"• Editor• Debugger• Documentatie	<ul style="list-style-type: none">• Make• Eclipse• Doxygen

4. Planning



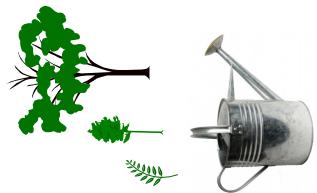
2

2. Betrouwbaarheid

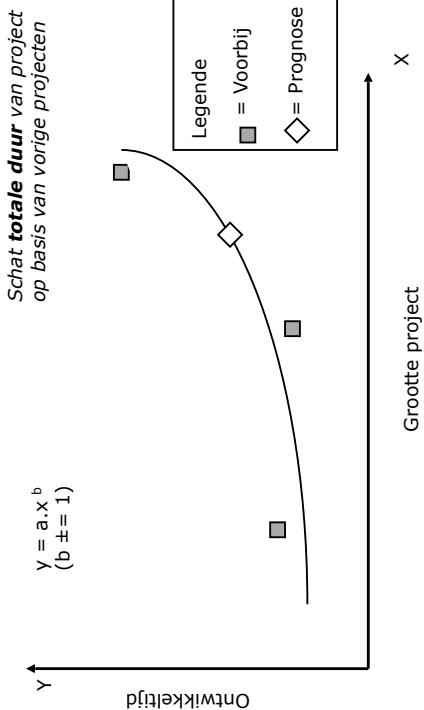
4. Planning

- Empirisch gespendeerde tijd voorbij projecten dient als basis voor schatting volgende

3

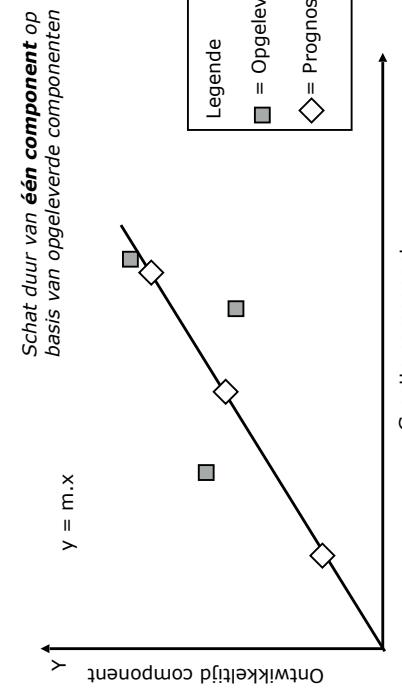


Empirische schatting (project)



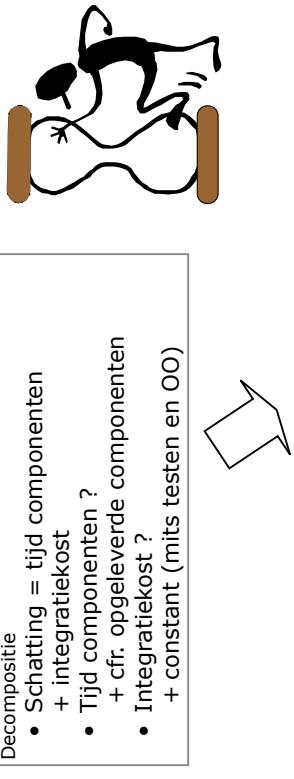
4. Planning

Empirische schatting (component)



4. Planning

Schatting door analogie



4. Planning

Grootte en Tijd

- $x =$ Grootte Component ? # stappen in use case + # uitzonderingen
- $y =$ Ontwikkelingstijd ? (*Zie tijdsbladen*)

vergelijking benaderende rechte
 $y = m \cdot x$

- Na oplevering n componenten: $(x_n, y_n) \Rightarrow m = \sum y_n / \sum x_n$
- Schatting y_{n+1} voor grootte $x_{n+1} \Rightarrow y_{n+1} = m \cdot x_{n+1}$

4. Planning

7

8

Vuistregel



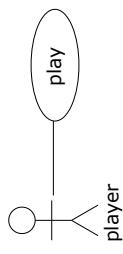
Empirische schatten is de basis voor een realistische planning.

- Waarom ?
- Betere controle over aanpassingen aan de planning
 - Hou tijdsbladen nauwkeurig bij
 - Maak prognose op basis van gespendeerde werk in het verleden

4. Planning

9

Use Case Grootte



Use Case 1: play

• ...

Steps

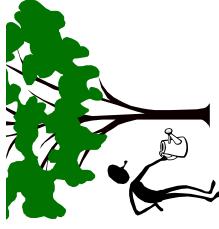
- 1. Two players start up a game (First is "O"; other is "X")
- 2. WHILE game not done
 - + 2.1 Current player makes move
 - + 2.2 Switch current player
- 3. Announce winner

$$\frac{\# \text{stappen}}{\# \text{uitzond.}} = \frac{5}{1} \\ \text{grootte} = 6$$

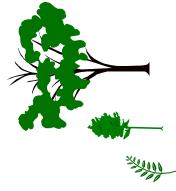
10

Voorbeelden

Zie voorbeelden in PlanTmp120 & PlanTmp121



Conclusie



- Betrouwbare prognoses zijn belangrijk
 - + Hou tijdsbladen nauwkeurig bij !
- Schatten impliceert fouten
 - + Voorzie een redelijke marge
 - + Vertrouw niet blindelings op de cijfertjes

4. Planning

11

2. Betrouwbaarheid

12

TicTacToe17

```
class TicTacToe {
    ...
    ...  

    char _winner;
};

TicTacToe::TicTacToe() {
    ...
    _winner = ' ';
    ENSURE(propertyInitialized(), "constructor ...");
}

void TicTacToe::writeOn(std::ostream& onStream) {
    ...
    onStream << "TicTacToe " << this->mMoves();
    << " - winner = " << this->getWinner() << endl;
    ...
}
```

4. Planning 13

TicTacToe18

```
TicTacToe() {
    ...
    setMoves(oMoves, xMoves: STRING)
    getMark (col, row: CHAR); CHAR
    setMark (col, row, marker: CHAR); CHAR
    notDone (): BOOLEAN
    nrOfMoves(): INTEGER
    doMove ()
    writeOn(o: ostream)
    getWinner (): CHAR
    reset()
}
```



Testscenarios + klasse onder test gaan hand in hand

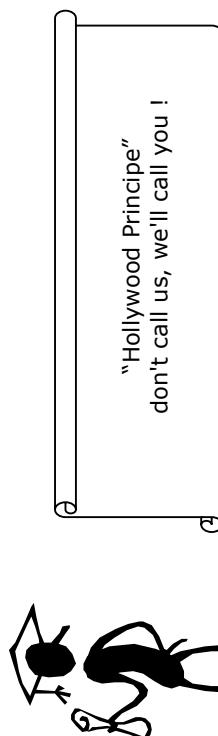
• + nieuwe testen voor getWinner

...
 onStream << "TicTacToe " << this->mMoves();
 << " - winner = " << this->getWinner() << endl;

Eenvoudig testen van getWinner
 ⇒ reset functionaliteit

4. Planning 14

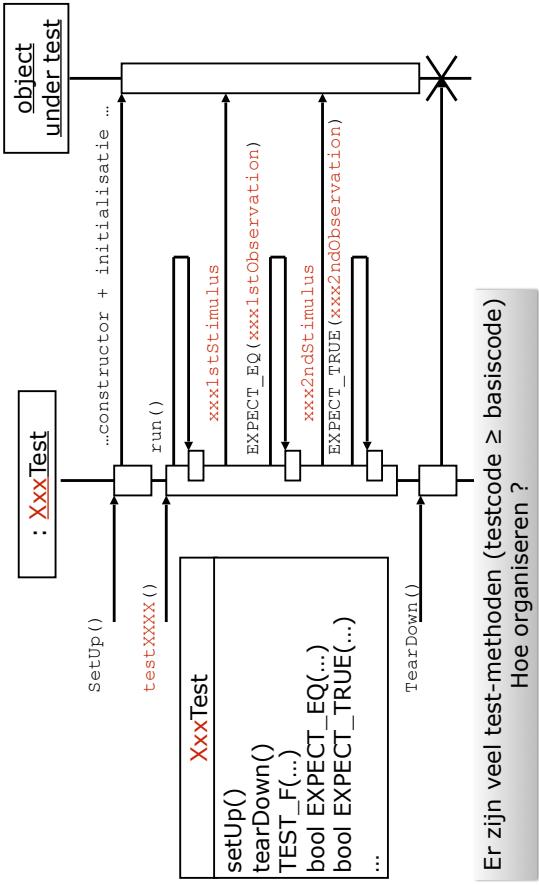
Vuistregel



- Waarom ?
 • Uitbreidung van bibliotheken via subklassen
 Hoe ?
 • Superklasse in bibliotheek legt protocol van oproepen vast
 • Subklassen kunnen gedrag uitbreiden
 + Voorbeeld: UnitTest (mindere mate TinyXML)

4. Planning 15

Generisch Unittest Protocol



4. Planning 16

Vuistregel



“3 lagen architectuur”

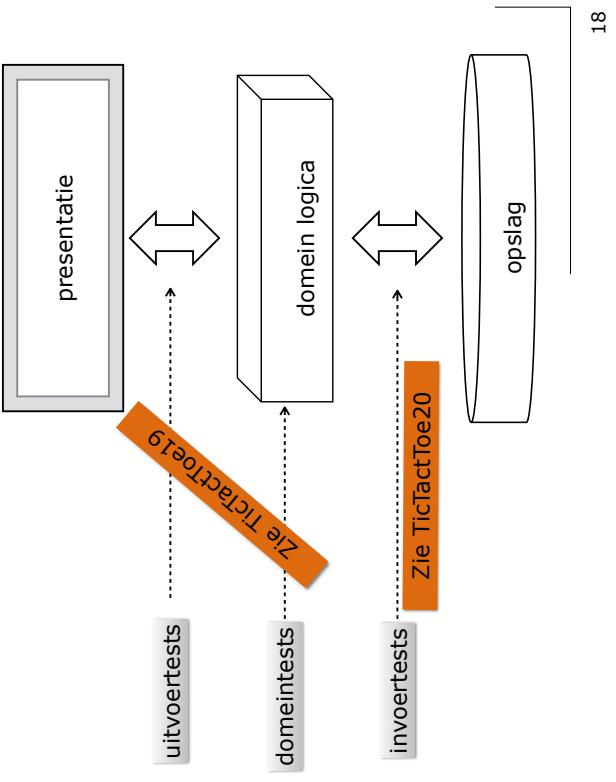
aparte componenten (en tests !) voor
(a) presentatie,
(b) domein logica,
(c) data-opslag

Waarom ?

- lokaal effect van veranderingen
- Hoe ?
 - Domeinmodel hangt niet af van databank, noch user-interface
- ⇒ Aparte tests voor invoer / uitvoer / domein

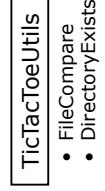
4. Planning

3-lagen architectuur

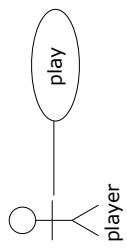


18

TicTacToe19



4. Planning



Use Case 1: play

• ...

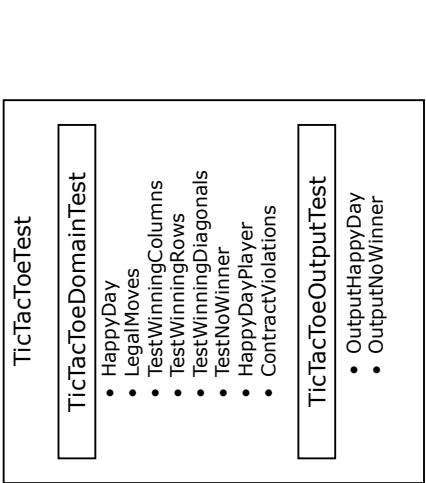
- 1. Two players start up a game (First is "O"; other is "X")
- 2. WHILE game not done
 - + 2.1 Current player makes move
 - + 2.2 Switch current player

• 3. Announce winner

- 2.1. [Illegal Move] System issues a warning

=> continue from step 2.1

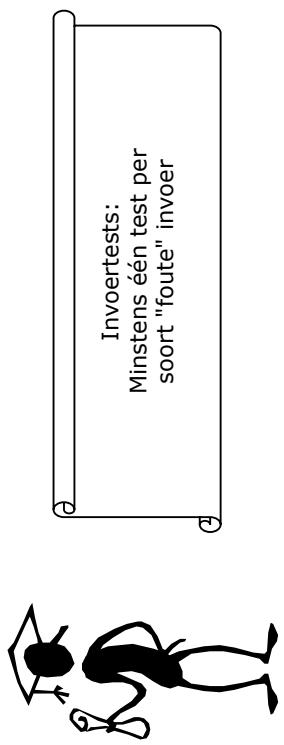
4. Planning



19

20

Vuistregel



Invoertests:
Minstens één test per soort "foute" invoer

Waarom ?
• Alle scenarios in de specificaties moeten getest worden

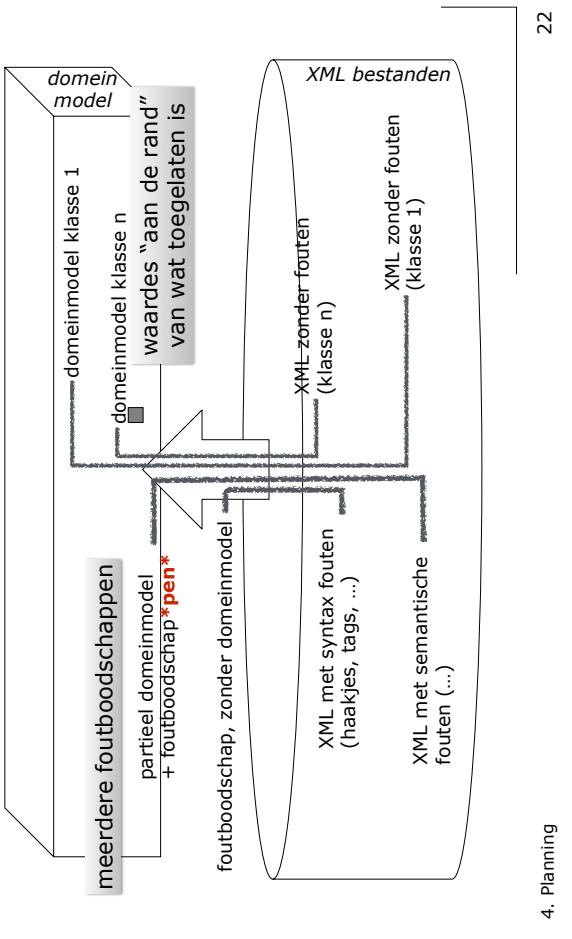
• Hoe ?

Controleer resultaat uitzondering via "EXPECT_EQ" ...
⇒ denk eraan: " Een test produceert zo weinig mogelijk uitvoer"

4. Planning

21

TicTacToe20



Zie TicTacToe20

Invoertests

TicTacToe: Specificatie

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



TicTacToeTest

TicTacToeOutputTest

TicTacToeDomainTest

TicTacToeInputTest

TicTacToeImporter

enum SuccessEnum {ImportAborted,
PartialImport, Success};
...
static SuccessEnum importTicTacToeGame (const char * inputfilename,
std::ostream& errStream,
TicTacToe& game);
• InputHappyDay
• InputLegalGames
• InputXMLSyntaxErrors
• InputIllegalGames

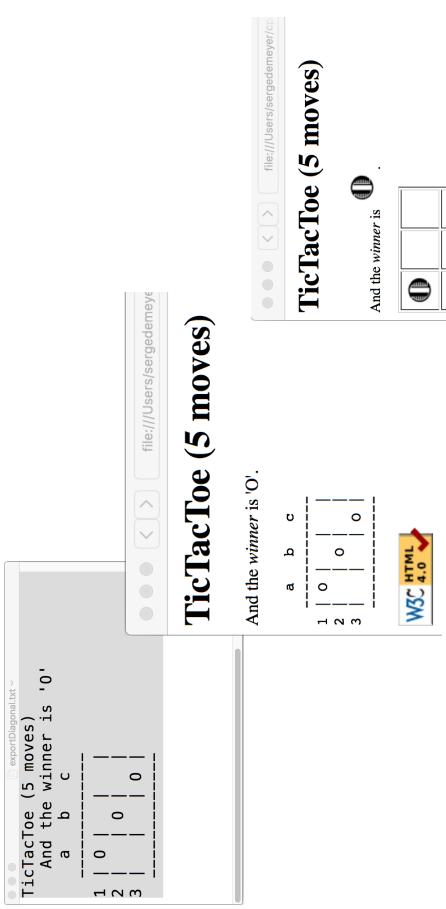
Variante: simple 2D en 3D
• XML met
• HTML met
• ...

23

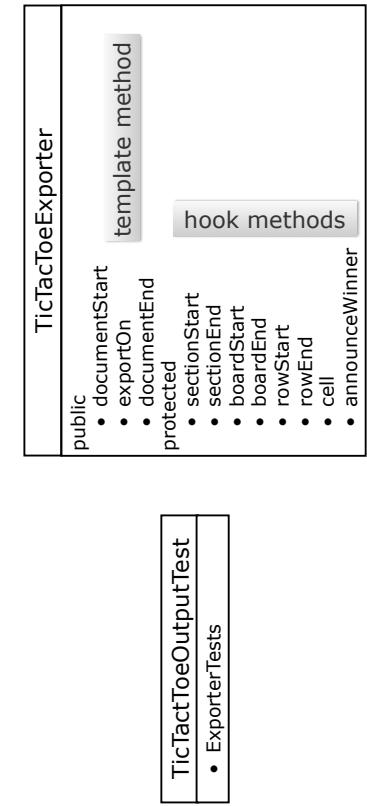
2. Betrouwbaarheid

24

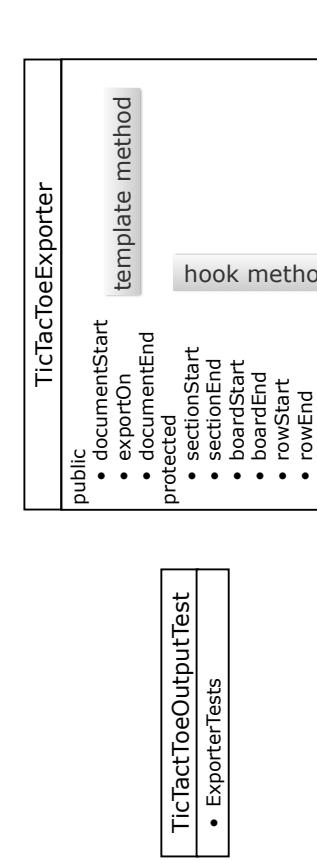
Three Variants of Output



4. Planning



26



27

exportOn = Template Method

```
void TicTacToeExporter::exportOn (std::ostream& onStream, TicTacToe &game) {
    char col, row;
    REQUIRE (this->properlyInitialized(), "...");
    REQUIRE (game.properlyInitialized(), "...");
    REQUIRE (this->documentStarted(), "...");

    if (game.nrOfMoves() == 1) {
        this->sectionStart(onStream, "TicTacToe (1 move)");
    } else {
        this->sectionStart(onStream, "TicTacToe (" +
            std::to_string(game.nrOfMoves()) + " moves)");
    }
    if (game.getWinner() != ' ') { this->announceWinner(onStream, game.getWinner()); }

    this->boardStart(onStream);
    for (row = minRow; row <= maxRow; row++) {
        this->rowStart(onStream, row - minRow + 1);
        for (col = minCol; col <= maxCol; col++) {
            this->cell(onStream, game.getMark(col, row));
        }
        this->rowEnd(onStream);
    }
    this->boardEnd(onStream);
    this->sectionEnd(onStream);
}
```

cell = Hook Method

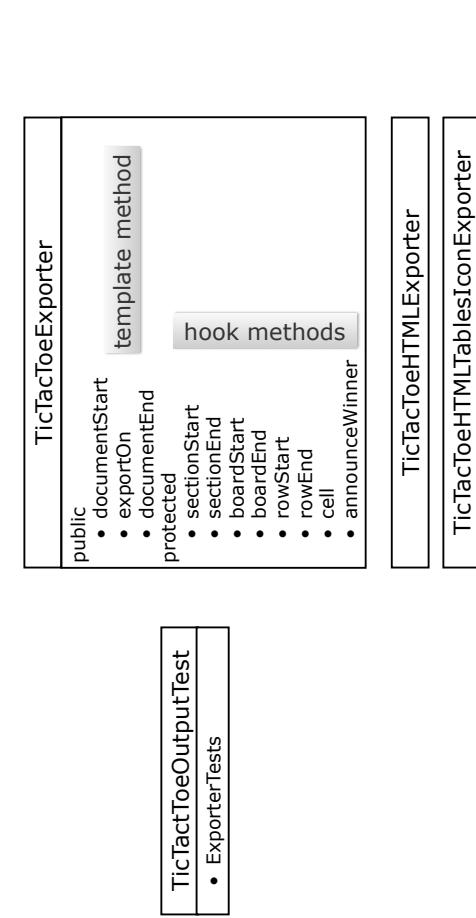
```
void TicTacToeExporter::cell
    (std::ostream& onStream, char const cellMarker) {
    onStream << " | " << cellMarker;

    void TicTacToeHTMLExporter::cell
        (std::ostream& onStream, char const cellMarker) {
        onStream << " | " << cellMarker;

        void TicTacToeHTMLTablesIconExporter
            (std::ostream& onStream, char const cellMarker) {
            onStream << " | " << cellMarker;
        }
    }
}
```

28

Hollywood Principle Revisited – TicTacToe21



29

4. Planning

27

4. Planning

Vuistregel

Contracten & Tests "List" (Double Linked) ?

```
list()  
  
    class node  
    {  
        public:  
            int value;  
            node *next;  
            node *prev;  
    };  
  
    class list  
    {  
        public:  
            node *front;  
            node *back;  
    };  
  
    void insertFront(int value);  
    void insertBack(int value);  
    void removeFront();  
    void removeBack();  
    void insertBefore(int value,  
                      node *nodeB);  
    void insertAfter(int value,  
                     node *nodeA);  
    void removeBefore(node *nodeB);  
    void removeAfter(node *nodeA);  
    void removeNode(node *newNode);  
    void printListFront();  
    void printListBack();  
};
```



Klassen die vaak gebruikt worden
hebben precieze contracten

... en veel tests !

Waarom ?

- Betere betrouwbaarheid door precieze beschrijving interface
- Hoe ?
- Leg de "normale" volgorde van oproepen vast
- Specificeer volgorde via de respectieve pre- en postcondities
- Specificeer volgorde via de volgorde (pre- en post-condities) verifiëren

4. Planning

29

Contracten ? Tests ?
Moeilijk want interface zonder predicate

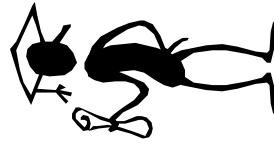
30

"List" met predicaten

```
//ENSURE(proprietyInitialized(), "constructor must end in properlyInitialized state");  
list()  
  
//REQUIRE(this->properlyInitialized(), "list wasn't initialized when calling includes");  
includes (int value): BOOLEAN;  
  
//REQUIRE(this->properlyInitialized(), "list wasn't initialized when calling insert");  
//REQUIRE(~includes(int), "before insert the list should NOT include the inserted value");  
//ENSURE(includes(int), "after insert the list should include the inserted value");  
insert (int value);  
  
//REQUIRE(this->properlyInitialized(), "list wasn't initialized when calling delete");  
//REQUIRE(includes(int), "before delete the list should include the inserted value");  
//ENSURE(~includes(int), "after insert the list should NOT include the inserted value");  
delete (int value);
```

Waarom ?
• Betere betrouwbaarheid door eenvoudige contracten
Hoe ?
• Specificeer predicate voor hoofdfunctionaliteit component
• Roep predicate op in pre- en post-condities

Contracten ? Tests ?
Makkelijker want interface met veel predicate



Prefereer een interface met predicate

Vuistregel

//ENSURE(proprietyInitialized(), "constructor must end in properlyInitialized state");
list()

```
//REQUIRE(this->properlyInitialized(), "list wasn't initialized when calling includes");  
//REQUIRE(~includes(int), "before insert the list should NOT include the inserted value");  
//ENSURE(includes(int), "after insert the list should include the inserted value");  
insert (int value);  
  
//REQUIRE(this->properlyInitialized(), "list wasn't initialized when calling delete");  
//REQUIRE(includes(int), "before delete the list should include the inserted value");  
//ENSURE(~includes(int), "after insert the list should NOT include the inserted value");  
delete (int value);
```

4. Planning

31

32

Vuistregels

- Betrouwbaarheid
 - Invoertests: Minstens één test per soort "route" invoer
 - Klassen die vaak gebruikt worden: precieze contracten + veel tests
 - Prefereer een interface met predicaten

- Ontwerpen
- "Hollywood Principle" – wij roepen jouw op als we je nodig hebben
 - + Template & Hook Methods

- "3 lagen architectuur"
- + aparte componenten voor (a) presentatie,
(b) domein logica, (c) data-opslag

- Plannen
- Empirische schatten is de basis voor een realistische planning

4. Planning

33

Evaluatie Criteria (Betrouwbaarheid)

Inleiding Software Engineering (2017 - 20??)

Student: student 1 - Student2: student3 - Student3: student3

Commentaar:

Tests

- O Nr aanwezig (= geen tests)
- O Vrkrd. gebruik (= manueel test)
- O Bepakt (= mt. alles getest)
- O Volledige (= geen uitvoer, verkeerde invoer)

Commentaar:

Contracten

- O Nr aanwezig (= geen assert)
- O Vrkrd. gebruik (=geen pre-post)
- O Bepakt (= alleen NUL)
- O Volledige (= allemaal goed)

Commentaar:

Invoertests

- O Excellent (= goede resultaten)
- O Goed (= voldoende resultaten)
- O Voldoende (= voldoende resultaten)
- O Slecht (= slechte resultaten)

Commentaar:

Klasseren

- O Aanwezig (= o.a. lijsten)
- O Bepakt (= vaak herbruikt)
- O Volledige (= allemaal goed)

Commentaar:

Prefererer een interface met contracten & worden

4. Planning

34

Evaluatie Criteria (Planning & Ontwerp)

Empirische schatten voor een realistische planning

Plan

O Niet aanwezig O Verkeerd gehanteert O Bepakt O Volledig O Goed O Excellent

Functionaliteit: O Onvoldoende O Minder dan behoefd O Zoals behoefd

I. Inlezen ... 2. Uitvoer

	VERPLICHT	VERPLICHT	VERPLICHT
1.1. Inlezen ...	2.1. ... wegschrijven	2.1. ... wegschrijven	2.1. ... wegschrijven
1.2. Inlezen ...	2.2. ... wegschrijven	2.2. ... wegschrijven	2.2. ... wegschrijven
1.3. Inlezen ...	2.3. ... acties wegschrijven	2.3. ... acties wegschrijven	2.3. ... acties wegschrijven

Belangrijk

VERPLICHT

Objectgericht Ontwerp (Alles slank/ruisen)

- O Geen
- O Inheritance
- O Reuse
- O Lijsten

Organisatie

- "Hollywood van componenten & tests"
- "3 lagen architectuur template methods"

Technieken

- Betrouwbaarheid
- Aanpassbaarheid
- Planning
- "Build"

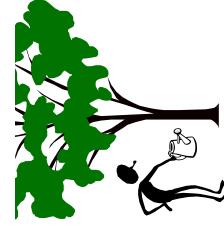
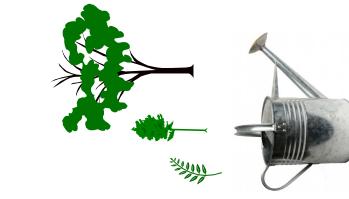
Werk具igen

- Editor
- Debugger
- Documentatie

4. Planning

35

Vereisten



Vereisten	Technieken
• Betrouwbaarheid	• Testen + Contracten
• Aanpassbaarheid	• Objectgericht ontwerp
• Planning	• Tijdsschatting

36



Gefundeerde beslissingen
 Green reden intutie
 Green reden doordacht

Algemeen
Behoersing werktuigen
Samenwerking:
Commentaar:



- VERPLICHT**
- gtest
 - DesignByContract.h
 - make

- BONUS**
- eclipse
 - doxygen

Project Software Engineering (BIJLAGEN)

Cursus

1ste Bachelor Informatica
Academiejaar 2015-2016

© Serge Demeyer – Universiteit Antwerpen



Universiteit Antwerpen

Context:	Inleiding Software Engineering
Project:	Project BA1 informatica (*)
Documentsoort:	Tijdsblad
Versie:	1.0 (*)
Datum:	18 maart 2011 (*)
Auteur:	Jan Jansens & Pieter Pietersen (*)
Status:	Voorbeeld (*)

Id ⁽⁺⁾	Behoefte ⁽⁺⁾				TOTAAAL ⁽⁺⁾	
		Initialen ⁽⁺⁾	Datum ⁽⁺⁾	Van ⁽⁺⁾	Tot ⁽⁺⁾	Totaal ⁽⁺⁾
1,1	Kamers en gangen inlezen				11:30	
	JJ	9-Feb		11:00	16:00	05:00
	PP	9-Feb		11:00	16:00	05:00
	PP	9-Feb		22:00	23:30	01:30
2,1	Simpele uitvoer				10:00	
	JJ	16-Feb		11:00	16:00	05:00
	PP	16-Feb		11:00	16:00	05:00

(*) Aan te passen

(+) Kolommen aan te passen

Context:	Inleiding Software Engineering
Project:	Project BA1 informatica (*)
Documentsoort:	Plan
Versie:	2.0 (*)
Datum:	18 maart 2011 (*)
Auteur:	Jan Jansens & Pieter Pietersen (*)
Status:	Voorbeeld (*)

Totaal tijdsbudget(*)	240,00
Totaal gespendeerd ^(*)	76,50
Totaal geschat 2.0 ^(*)	140,25
Totaal geschat 2.1 ^(*)	51,00
SALDO	-27,75

Behoeftie Id⁽⁺⁾
OPGELEVERD 1.0
1,1
2,1

$$\text{sum}(Gespendeerd) / \text{sum}(\# Stappen) = 76.5 / 18$$

OP TE LEVEREN 2.0

22-Apr

$$\begin{array}{r}
 y = m * x \\
 9 \quad \quad \quad 38,25 \\
 10 \quad \quad \quad 42,5 \\
 10 \quad \quad \quad 42,5 \\
 \hline
 4 \quad \quad \quad 17 \\
 \hline
 & 140,25
 \end{array}$$

4,25 << berekenen

$$\begin{array}{r}
 y = m * x \\
 \hline
 9 & 38,25 \\
 10 & 42,5 \\
 \hline
 4 & 17 \\
 \hline
 & 140,25
 \end{array}$$

REN 2.1

$y = m/x$	8,5	<< schrappen
12,75	8,5	<< schrappen
	21,25	<< schrappen
		51

$$y = m * x$$

x

27-Ma

nuttig
nuttig
nuttig
nuttig

- (*) Aan te passen
- (+) Kolommen aan te passen

Context:	Inleiding Software Engineering
Project:	Project BA1 informatica (*)
Documentsoort:	Plan
Versie:	2.1 (*)
Datum:	27 mei 2011 (*)
Auteur:	Jan Jansens & Pieter Pietersen (*)
Status:	Voorbeeld (*)

Total tijdsbudget (*)	240,00
Total gespendeerd (*)	226,00
Total geschat 2.1 (*)	24,04
<i>SALDO</i>	-10,04

<< overwerk ?

Behoefte Id ⁽⁺⁾	Behoefte	Prioriteit ⁽⁺⁾	oplevering ⁽⁺⁾	#stappen ⁽⁺⁾	geschat ⁽⁺⁾	TJD gespendeerd ⁽⁺⁾
OPGELEVERD 2.0						
1,1	...	verplicht	18-Mar	12	nihil	53,50
2,1	...	verplicht	18-Mar	6	nihil	23,00
1,2	...	verplicht	22-Apr	9	38,25	36,00
1,4	...	belangrijk	22-Apr	10	42,50	40,00
1,5	...	belangrijk	22-Apr	10	42,50	73,50 << pech gehad
				47		226,00
<i>sum(Gespendeerd) / sum(# Stappen)</i>		= 226 / 47		<i>m =</i>	<i>4,81 << herberekenen</i>	
OP TE LEVEREN 2.1						
2,2	...	belangrijk	27-May	4	$y = m * x$	
1,3	...	nuttig	27-May	2	19,23	
1,6	...	nuttig	27-May	3	9,62	
2,3	...	nuttig	--niet	2	14,43 << schrappen ?	
2,4	...	nuttig	--niet	5	0,00	
						24,04

- (*) Aan te passen
- (+) Kolommen aan te passen