Discount method for programming language evaluation

Svetomir Kurtev and Tommy Aagaard Christensen

1 February - 28 June

Date	Svetomir Kurtev
Date	Tommy Aagaard Christensen

AALBORG UNIVERSITY
STUDENT REPORT

Department of Computer Science
Computer Science
Selma Lagerlöfs Vej 300
Telephone 99 40 99 40
Telefax 99 40 97 98
`http://cs.aau.dk`

Title:

Discount method for programming language evaluation

Project period:
1 February - 28 June

Project group:
dpt108f16

Participants:
Svetomir Kurtev
Tommy Aagaard Christensen

Supervisor:
Bent Thomsen

Abstract:

In methods for programming language design evaluation there is a gap between small internal methods and large scale surveys and studies. A similar gap in HCI has been filled by the discount usability evaluation method. In this report, the discount usability methods applicability on programming languages was examined, and it was found usable but better suited for compiler and IDE evaluation over language design evaluation. To create a method to fill the gap, a modified version of the usability method, where the IDE and compiler was removed, was tested.

Pages: 5

Appendices: 6

Copies: 0

Finished: 13 June, 2016

Preface

The following report was written by Svetomir Kurtev and Tommy Aagaard Christensen in accordance with the conclusion of the tenth and final semester of the Computer Science Master Program at Aalborg University.

We would like to thank Bent Thomsen for the help and guidance he provided us with throughout the development of the project.

Contents

Preface **i**

0.1 Tasks 1

0.2 Samples 2

0.3 Interview 2

I Bibliography **4**

Bibliography **5**

Part I

Bibliography

Chapter 1

List of Abbreviations

PLATEAU	Evaluation and Usability of Programming Languages and Tools
HCI	Human-Computer Interaction
UK	United Kingdom
USA	United States of America
FPL	First Programming Language
TBB	Threading Building Blocks
IDE	Integrated Development Environment
API	Application Programming Interface
IDA	Instant Data Analysis
VDA	Video Data Analysis
RPG	Role-Playing Game
DVR	Digital Video Recorder
OO	Object-Oriented

Chapter 2

Task Sheet

Task 1:
Imagine a simple supermarket billing system which can specify orders and calculate the total price of ordered items. For the sake of simplicity, we work with oranges and bananas as our products. Oranges cost 5$ per piece and bananas 4$ per piece, respectively. Create a system that:

- Can calculate the total price given a number of oranges and bananas bought.
- Adds a different price for buying a specific amount of an item
- Make triplets of oranges cost 10$ in total instead of 15$
- Make 5 bananas cost 10$ instead of 20$
- Adds a discount of 10% to the total price for regular customers

Task 2:
Imagine you have 2 football teams and each team has an equal amount of players. Each player has both his first and last name written down as well as their age. Try to find the following things:

- 2 or more players with the same first or last name in the same team
- 2 or more players with the same first or last name across the two teams
- 2 or more players with the same first name and age in the same team

Task 3:
Imagine you have a simple Role playing game. You have a base character which can be specialized in different classes such as Warrior, Mage etc. Every character has a certain amount of hitpoints and has the ability to attack other characters.

- Create a system for characters who all have:

  - Hit points and the ability to replenish them
  - The ability to attack other characters

- Allow a character to have a specific class
- Add a specific unique resource to every class (Warriors get fury, Mages get mana)
- Add a special unique attack to every class (Warriors get “Execute”, Mages get “Fireball” etc.)
  - These unique attacks spend the unique resource, respectively (e.g. Fireball costs 10 mana)
- Add the ability for every class to replenish their unique resource.

Task 4:
For some given text (for example your full name), write a procedure which:

- Prints the text in reverse order
- Prints the letters from the text in an alphabetical order
- Finds if there are duplicate letters in the text and if there are, list how many are duplicated (e.g. “Tommy” will give the result of 1, while “Christensen” has 3)

Chapter 3

Sample Sheet

General information & code examples

Quorum is an evidence-based programming language, designed from the outset to be easily understood and picked up by beginners. One of the design decisions taken includes the full omit of brackets when defining scopes. Keywords in the language make use of a more natural mapping to the real world, such as "text" for strings, "number" for doubles and floats and "repeat" for loops. Conditional statements such as if-statement are always ended with the keyword "end" which specifies the end of scope.

Data types

```
integer a = 5
number b = 10.2
text c = "John"
boolean d = true
```

Type conversion:

```
text someText = "5.7"

number someNumber = cast (number, someText)
```

Simple operation with arrays and conditional statements

The following code creates an array a with some randomly placed elements. It then sorts the array and iterates through the array to create an output with all the elements.

```
use Libraries.Containers.Array
action Main
        text unordered = "fdebaac"
        Array<text> a = unordered:Split("")
        a:Sort()
        integer i = 0
        text out = ""
        repeat while i < a:GetSize()
                out = out + a:Get(i) + ";"
                i = i + 1
        end
        output out
end
```

Output is: `a;a;b;c;d;e;f;`

This is an example of an action using if- else statements

```
action checkIntervals(integer i)
    if i < 10
        output "it is less than 10"
    elseif i = 10 or i > 10 and i <= 15
        output "it is less than or equal to 15 but greater or equal to 10"
    else
        output "it is greater than 15"
    end
end
```

Classes & Inheritance

To demonstrate classes and inheritance in quorum, we use the example of the animal family felidae and a cat belonging to that family:

First the superclass felidae looks like this:

```
class felidae
    text name = "Sebastian"

    public action Paws() returns integer
        return 4
    end

    action Purr()
        output name + ": rhrhrhrhrhrhrhrhrhrhrhrh"
    end
end
```

We then create the cat subclass like this:

```
class cat is felidae
    action Meow
        output parent:felidae:name + ": meow"
    end
end
```

To show the code in action we then use a main action that looks like this:

```
action Main
    cat sampleCat
    sampleCat:Purr()
    sampleCat:Meow()
    output sampleCat:Paws()
end
```

Where we instantiate a cat and call both the action from the superclass and the subclass giving the output of:

```
Sebastian: rhrhrhrhrhrhrhrhrhrhrhrh
Sebastian: meow
4
```

Worth noting is that we need to specify that the action Paws is public before we can call it from outside the class since it returns something (actions that does not return something are public by default). Likewise, if we in main where to write something like:

```
output sampleCat:parent:felidae:name
```

In order to access the name property, it would give an error since the name is not public.

Chapter 4

Interview Questions

1. What do you think about the language? Was it easy to learn?
2. Did you find some of the design odd?
3. How does Quorum relate to other languages you have experience in?
4. How did you find the tasks? Were they too challenging or too easy?
5. What do you think about coding without a compiler?

Chapter 5

Interview notes

As previously mentioned, the interviews with the participants were recorded in order to preserve the necessary feedback which helped in analysing the results. Instead of providing the entirely of the interviews in the form of transcripts, we decided to condense the information in key points instead. This helped us to analyse the data from the interviews much easier and find out how many occurrences of a given problem there are across all the participants. Additionally, this section encapsulates the essential parts of each interview and highlights what every participant had to say in terms of feedback, suggestions for further improvements and encountered problems.

5.1 Participant #1 (Pilot Test)

- Thought that using colon (`:`) instead of dot (`.`) was weird, both because it goes against the norm (the participant had experience with several languages which use the dot notation) and because dot is easier to type.
- Thought that tasks are trivial to understand but take time to code
- Task 1 was too broad in the definition causing the task to be too large and time-consuming and the participant to spend time on unintended things.
- Mentioned that although repetitive to a certain extend, task 2 was tricky and very good at conveying that you have to pay attention when copying code. (*He actually fell into that trap and he did not realise it up until the facilitator intervened and pointed that out.)
- Thought that tasks 2 and 4 were quite good in terms of their intended purposes while task 3 (operation on strings) was trivial and very similar to task 2.
- He found the samples of conditional use not being able to clearly convey the differences between Quorum and other known languages he had experience in. In particular == vs. = and `and` vs `&&` did not stand out.

- Thought that it was good that the sample sheet was split in categories to make it easier for the facilitator to reference them when asked.

5.2 Participant #2

- Thought that a lot of the notations were unintuitive because they differed from the mathematical norm
- Found the keyword `action` confusing
- Thought that using `loop` would be easier than `repeat`
- It was difficult for him to devise the code needed for solving the tasks, although he found the mathematics behind quite easy
- It was daunting to not have any fallback or assistance when trying to code and learn how to code without a compiler

5.3 Participant #3

- Suggested that we should add specific values for task 3
- Wondered how to define return types of an action
- Quorum does not have parameterized constructors
- Suggested that we add how to get the size of an array with an in-build action
- Forgot to increment loop counters
- Forgot to add the `repeat` keyword
- Thought that Quorum has a limited number of looping constructs, but it is easy to learn, write and understand
- Quorum is very terse
- Thought that `output` makes more sense than using `print`
- Thought that `returns` of an action seems intuitive
- Liked the `is` keyword for class inheritance
- Thought that Quorum is similar to C, with a different syntax (programmer-friendly C)
- Thought that the lack of parameterized constructors is not that limiting, but does not have enough experience to say with a certainty

- Found the tasks not too challenging
- Thought that not using a compiler is not much of a hindrance
- Found the example sheet indispensable and very helpful
- Suggested that we could add more examples for loping constructs

5.4 Participant #4

- Found it strange to use words as a means of closing scopes instead of brackets as well as using colons instead of dots
- Thought the languages is straightforward and easy to use
- Used a `float` instead of a `number` keyword, as well as `string` instead of `text`
- Forgot to add `returns` keyword at the end of an action
- Forgot to increment the counters on loops
- Had some problems with scoping by making use of the `end` keyword
- Found the tasks specific, understandable and clear
- Thought that `.` makes more sense than `:`
- Suggested that we add an example of method inheritance on the example sheet
- Suggested that we change the / on task 2 with an "or"
- Suggested that we add a sort action on the example sheet constructs

5.5 Participant #5

- Found Quorum is similar to C
- Quorum has similar design to other languages `string` instead of `text`
- Tended to over-complicate things and thus - over-engineer the tasks
- Made use of the example sheet quite frequently
- Coding without a compiler was unpleasant and felt like being in an exam, unable to get a feedback from what's being written down (Does not allow a great deal of experimentation)
- Had difficulties with the syntax of arrays - using the `[]` notation instead of the `get(i)` inbuilt method

- Forgot to write the import for using arrays, as specified on the example sheet
- Found the tasks very good at conveying our intended purposes and easy to understand
- Found the amount of tasks good and reasonable
- Found Task 3 to be a bit tricky since you have to specifically think in terms of inheritance from the start
- Found the example sheet informative and referred to it several times

5.6 Participant #6

- Quorum's design seems a bit confusing
  - Closing the scopes of If-statements with `end`
  - Lack of parameterized constructors
  - Lack of a for-loop
- Found the tasks very good and the example sheet - very concise
- Found coding without a compiler scary without "the safety net"
- Typed = instead of == for an inequality operator
- Thought it might be more intuitive to use a Get method directly compared to how it is being used in the language
- Typed `.` instead of `:`
- Thought that ending classes with something different than the `end` keyword will make more sense
- Found closing the scope of if-statements with the `end` keyword confusing and said that brackets would make it more readable (similarly to OO languages such as Java and C#)
- Found Quorum less verbose than other OO languages
- Forgot to increment the counter variable outside of a loop
- Although the participant over complicated the tasks based on the provided description, he found them very good and efficient at what they try to convey
  - Task 2 - the description of the task seems rather confusing, which made the participant to over-engineer the solution
  - Task 3 - doable
- Found the example sheet contains enough content in order to solve the tasks
- Had a few suggestions how to improve the overall look of the example sheet

5.7 Participant #7

- Found Quorum similar to Pascal and C#
- Liked certain parts of the language and disliked others
- Found the use of `repeat` unnecessary since it does not make sense in conjunction with the standard loop wording
- Noticed that you have to close a class/action with an `end` keyword
- Suggested that implicit type casting would be better for novices
- `:` used in different scenarios might be confusing
- Thought that the `returns` keyword can have a better placement in the action's signature
- Noticed that you have to use a library for an array
- Said that the `end` keyword does not make much sense and rather see a `begin-end` scoping construct, similar to Pascal and Python - only indentation
- Casting data types could be dangerous for novices
- Found the `returns` keyword's placement not so intuitive
- Found the `end` keyword for if-statements not so adequate, can use indentation instead similar to Python
- Found the tasks very good:
  - Task 2's challenge of reusing code is a good exercise
  - Task 2 could have a 2 predefined lists with names
- Said that the task encompass a good portion of constructs
- Suggested we could add a setup for easier start with the tasks
- Suggested we give better titles on the examples sheet and better indexing when looking for things
- Coding without a compiler did not matter that much in his opinion
- Found it great that the facilitator could say if the task is done or not
- `GetSize()` and `Add()` in-build methods examples were missing
- Acknowledged that the code samples are highlighted and there are working examples
- Said that we should be consistent with the working titles

5.8 Participant #8

- Found Quorum intuitive to use, but limited in terms of available constructs
- Suggested that `returns nothing` would be intuitive
- Found the naming of keywords inconsistent (Arrays with capital A and everything else with small letters)
- Found it confusing not to use indentation for scopes
- Found the lack of semicolons a very good thing
- Liked the `is` keyword for class inheritance
- Pointed out the lack of a `continue` construct for loops
- Would have liked more features from functional programming
- Suggested we could make the “or” and “and” statements bolded in task 2
- Noticed the lack of an aggregate += operator
- Quorum reminds him of OO languages and similar to Python
- Would have liked a summary of all the examples on the examples sheet
- Found the examples not so sufficient per task
- Suggested that we could highlight important parts on the task sheet
- Found the lack of a compiler while coding “dangerously scary”
- Over-engineered task 1
- Suggested that we could have an additional sheet with solutions to the tasks
- Separate each task on a separate paper so it is easier to navigate

Chapter 6

Participant code

6.1 Participant #1

```
use Libraies.Container.Array
use public

action Main
Array<Cucumber> a
a:add(Cucumber c1)
  a:add(Cucumber c2)
  a:add(Cucumber c3)
  a:add(Cucumber c4)
CalcTotal(a)
end


action CalcTotal(Array<Cucumber> arr)
  number total = 0
  integer i = 0
  total = Cucumber.Price(arr.GetSize())
  output "total = " + total

end

Class Cucumber
  integer id
  number price
  number bulkPrice
  integer bulkCount
  number percentageDiscount
  boolean bORp


  public action Price(integer amount)
    integer remains = mod amount
    integer numdiscount = amount / bulkCount

    number value
    if bORp
      value = remains* price + numdiscount * bulkPrice
    else
      value = 100 - percentageDiscount * price *amount
    end

    return value
  end
end


//TASK2

Class Player
  public text FN
  public text LN
  public integer age

  action make (text first, text last, integer _age)
  FN = first
  LN = last
  age = _age
  end
end

action Main
Aray <Player> T1
Array <Player > T2

Player p1
Player p2
Player p3

Player p4
Player p5
Player p6

p1:make("a","b",10)
p2:make("a",zebra,1)
p3:make("gi", "joe", 65)
```

```

T1:add(p1)
T1:add(p2)
T1:add(p3)

p4:make("anotherguy","b",20)
p5:make("c","c",11)
p6:make("d","d",25)

T2:Add(p4)
T2:Add(p5)
T2:Add(p6)

  public action FindFFNLNbetweenTeams returns integer
    integer i = 0
    integer j = 0
    integer found = 0
    repeat while i < T1:GetSize()
      repeat while j < T2:GetSize()
         if T2:Get(j):FN = T1:Get(i):FN
          found = found +1
        else if T2:Get(j):LN = T1:Get(i):LN
          found = found +1
        end
      end
    end
    return found
  end

    public action FindFFNLNinTeam(Array<Player>) returns integer
    integer i = 0
    integer j = 0
    integer found = 0
    repeat while i < T:GetSize()
      repeat while j < T:GetSize()
         if T:Get(j):FN = T:Get(i):FN
          found = found +1
        else if T:Get(j):LN = T:Get(i):LN
          found = found +1
        end
      end
    end
    return found
  end

    public action FindAgebetweenTeams returns integer
```

```
      integer i = 0
      integer j = 0
      integer found = 0
      repeat while i < T1:GetSize()
        repeat while j < T2:GetSize()
           if T2:Get(j):Age = T1:Get(i):Age
            found = found +1
          end
        end
      end
      return found
    end

      public action FindAgeinTeam(Array<Player>) returns integer
      integer i = 0
      integer j = 0
      integer found = 0
      repeat while i < T:GetSize()
        repeat while j < T:GetSize()
           if T:Get(j):Age = T:Get(i):Age
            found = found +1
          end
        end
      end
      return found
    end


      public action FindAgeinTeam(Array<Player>) returns integer
      integer i = 0
      integer j = 0
      integer found = 0
      repeat while i < T:GetSize()
        repeat while j < T:GetSize()
           if T:Get(j):Age = T:Get(i):Age and T:Get(j):FN = T:Get(i):FN
            found = found +1
            j = j+1
          end
          i = i+1;
        end
      end
      return found
    end

  end
```

```

//TASK 4

Class Base

integer hp
integer dmg

  action do()
  end
  action Attack(Base target )
  target:takeDamage(dmg)
  end

  action takeDamage(integer damage)
  hp = hp - damage
    if hp >= 0
      kill()
    end
  end

  action replnishHP(integer amount)
  hp = hp + amount
  end

  action kill ()
  delete me
  end

end

Class Warior ia Base
hp = 150
dmg = 10;
integer fury = 100

  action do()
  fury = fury +1
  end

  action strongAttack
    if fury > 10
      target:takeDamage(dmg+10)
      fury = fury - 10
    else
    output "might knight whines like tiny baby men"
    end
  end
end

Class Mage is Base
hp = 70
damage = 12
integer mana

  action do()
  mana = mana +1
  end


  action heal (target)
    if mana > dmg
    target.replnishHP(dmg)
    mana = mana - dmg
    end
  end

end

Action Main
for each base
do()
end
end
```

6.2 Participant #2

```
action gettotal (integer Oranges,integer Bananas,boolean isregular)
    returns integer
integer total=0
total=total+Oranges*5+Bananas*4-5*Oranges/3-10*Bananas/5
if isregular = true
total=total*0.9
end
refurn total
end
output gettotal (3,5,true)

first name  last name  age  team
```

6.3 Participant #3

```
class Test1
  integer OrangePrice = 5
  integer BananaPrice = 4

  action TotalPrice(integer oranges, integer bananas)
    Output oranges * Orangeprice + bananas * BananaPrice
  end

  action TotalPriceWithDiscount(integer oranges, integer bananas, boolean
      regular)
    number result = 0
    repeat while oranges > 3
      result = result + 10
      oranges = oranges - 3
    end
    result = result + oranges * OrangePrice

    repeat while bananas > 5
      result = result + 10
      bananas = bananas - 5
    end
    result = result + bananas * BananaPrice

    if regular
      Output result * 0.9
    else
      Output result
    end
  end
end

class Test2
  Array<Player> Team1
  Array<Player> Team2

  action SameFirstLastNameSameTeam(Array<Player> team) returns boolean
    integer i = 0
    repeat while i < team.GetSize()
      text firstName = team:Get(i):FirstName
      text lastName = team:Get(i):LastName
      integer j = 0
      repeat while j < team.GetSize()
        if not(j == i) and (firstName == team:Get(i):FirstName or lastName
            == team:Get(i):LastName)
```

```
          return true
        end
      end
    end
    return false
  end

  action SameFirstLastNameDifferentTeams() returns boolean
    integer i = 0
    repeat while i < Team1.GetSize()
      text firstName = Team1:Get(i):FirstName
      text lastName = Team1:Get(i):LastName
      integer j = 0
      repeat while j < Team2.GetSize()
        if firstName == Team2:Get(i):FirstName or lastName ==
           Team2:Get(i):LastName
          return true
        end
      end
    end
    return false
  end

  action SameFirstLastNameSameTeam(Array<Player> team) returns boolean
    integer i = 0
    repeat while i < team.GetSize()
      text firstName = team:Get(i):FirstName
      integer age = team:Get(i):Age
      integer j = 0
      repeat while j < team.GetSize()
        if not(j == i) and (firstName == team:Get(i):FirstName and age ==
           team:Get(i):Age)
          return true
        end
      end
    end
    return false
  end
end

class Player
  text FirstName
  text LastName
  integer Age
end

```

```
class Character
  number Health

  action ReplenishHealth(integer amount)
    Health = Health + amount
  end

  action Attack(Character target)
    target:Health = target:Health - 10
  end
end

class Mage is Character
  number Mana

  action Fireball(Character target)
    if Mana >= 10
      Mana = Mana - 10
      target:Health = target:Health - 20
    end
  end

  action ReplenishMana(integer amoutn)
    Mana = Mana + amount
  end
end

class Warrior is Character
  number Fury

  action Execute(Character target)
    if Fury >= 25
      Fury = Fury - 25
      if target:Health < 30
        target:Health = 0
      else
        target:Health = Target:Health - 10
      end
    end
  end

  action ReplenishFury(integer amount)
    Fury = Fury + amount
  end
end

```

```
class Test4
  text Text = "Rasmus Moeller Jensen"
  Array<Text> a = Text:Split("")

  action PrintReverse()
    integer i = a:GetSize() - 1
    text Result = ""
    while i >= 0
      Result = Result + a:Get(i)
      i = i + 1
    end
    Output Result
  end

  action PrintAlphabetical()
    Array<Text> b = a
    b:Sort()
    text Result = ""
    integer i = 0
    repeat while i < b:GetSize()
      Result = Result + b:Get(i)
      i = i + 1
    end
    Output Result
  end

  action FindDuplicates()
    integer i = 0
    integer j = 0
    integer Result = 0
    Array<Text> AlreadyTested

    repeat while i < a:GetSize()
      j = 0
      repeat while j < a:GetSize()
        if not(i == j) and not(a:Get(i) == " ") and not(a:Get(j) == " ")
            and not(AlreadyTested:Contains(a:Get(i))) and a:Get(i) ==
            a:Get(j)
          Result = Result + 1
          AlreadyTested:Add(a:Get(i))
        end
        j = j + 1
      end
      i = i + 1
    end
    Output Result
  end
end
```

6.4 Participant #4

```
use Libraries.Containers.Array


action CalculatePrice(integar nBananas, number pBananas, integar nOranges,
    number pOranges, boolean regular) returns number
  number totalgroup = ((nBananas / 5) * 10) + ((nOranges / 3) * 10)
  number rBananasPrice = (nBananas % 5) * 4
  number rOrangesPrice = (nOranges % 3) * 5
  number total = totalgroup + rBananasPrice + rOrangesPrice
  if regular total = total * 0.9
  return total
end

//each array entry is a string with name, second name
action FindSameFirstNames(Array<text> teamone, Array<text> teamtwo)
    returns string
  Array<text> SameNames;
  integar i = 0
  integar j = 0
  repeat while i < teamone:GetSize()
    repeat while j < teamtwo:GetSize()
      string pAF = teamone.Get(i).Split(",").Get(0)
      string pBF = teamtwo.Get(j).Split(",").Get(0)
      string pAL = teamone.Get(i).Split(",").Get(1).Trim()
      string pBL = teamtwo.Get(j).Split(",").Get(1).Trim()
      if(pAL = pBL or pAF = pBF) return players.Get(i) + " : " +
          players.Get(j)
      i = i + 1
      j = j + 1
    end
  end
end

//each array entry is a string with name, second name, age
action FindSameFirstNamesAndAge(Array<text> teamone) returns string
  Array<text> SameNames;
  integar i = 0
  repeat while i < teamone:GetSize()
    integar j = 0
    repeat while j < teamone:GetSize()
      string pAF = teamone.Get(i).Split(",").Get(0)
      string pBF = teamone.Get(j).Split(",").Get(0)
      integar pAA = cast (integar, teamone.Get(i).Split(",").Get(2))
      integar pBA = cast (integar, teamone.Get(j).Split(",").Get(2))
      if(pAF = pBF and pAA = pBA) return players.Get(i) + " : " +
         players.Get(j)
      j = j + 1
    end
    i = i + 10
  end
end



class character
  public integar hp = 100
  public integar resourceAmount = 100

  public action Attack(character defender, integar amount)
    defender:hp = defender:hp - amount
  end

  public action Recover(integar amount)
    hp = hp + amount
  end

  public action RecoverResource(integar amount)
    resourceAmount = resourceAmount + amount
  end
end

class mage is character

  public string resourceName = Mana

  public action Fireball(character defender, integar amount)
    parent:character:Attack(defender,amount)
    parent:character:resourceAmount = parent:character:resourceAmount - 10
  end

end

class warrior is character

  public string resourceName = Rage

  public action Pummel(character defender, integar amount)
    parent:character:Attack(defender,amount)
    parent:character:resourceAmount = parent:character:resourceAmount - 20
  end

end

class taxAccountant is character

  public string resourceName = Money

  public action ChargeWithTaxEvation(character defender, integar amount)
    parent:character:Attack(defender,amount)
    parent:character:resourceAmount = parent:character:resourceAmount - 50
  end

end

public action ReverseText(text texttotreverse) returns text
  text out = ""
  Array<text> characters = texttotreverse:Split("")
  integar count = character:GetSize() - 1
  repeat while count >= 0
    out = out + characters:Get(count)
    count = count - 1
  end
  return out
end

public action SortCharacters(text string)
  Array<Text> characters = string:Split(""):Sort()
  integar count = character:GetSize()
  integar i = 0
  repeat while i < count
    output characters:Get(i)
    i = i + 1
  end
end

public action FindDuplicates(text string) returns integar

  Array<text> characters = string:Split("")
  Array<text> foundChar

  integar i = 0

  repeat while i < characters:GetSize()
    integar j = 0
    repeat while j < characters:GetSize()
      if characters:Get(i) = characters:Get(j)
        integar k = 0
        boolean found = false
        repeat while k < foundChar:GetSize()
          if characters:Get(i) = foundChar:Get(k)
            found = true
        end
        if not found
          foundChar:Add(characters:Get(i)

      j = j + 1
    end
    i = i + 1
  end

  return foundChar:GetSize()
end
```

6.5 Participant #5

```
action Main
        action calculateFruit(integer banana, integet orange) returns
            integer
                integer orangePrice = 5
                integer bananaPrice = 4

                return orange*orangePrice + banana*bananaPrice
        end

        action calculateFruit(integer banana, integet orange, boolean
            regular) returns integer
                integer orangePrice = 5
                integer bananaPrice = 4

                orangesDiscount = orange/3
                orangeRemainder = orange mod 3
                bananaDiscount = banana/5
                bananaRemainder = banana mod 3

                sum = orangeRemainder*orangePrice + orangesDiscount*10 +
                    bananaRemainder*bananaPrice + bananaDiscount*10

                if regular
                        return sum-sum*0.1
                else
                        return sum
                end
        end

        // firstname,lastnam

        // 0,firstname
        // 1,lastnam
        action findPlayers1(Array<text> team) returns Array<text>

                integer i = 0
                Array<Array<text>> players
                repeat while i < team:GetSize()
                        Array<text> player = team:Get(i).Split(",")
                        players:Add(player)
                        i = i + 1
                end

                integer i = 0
                integer j = 0
                repeat while i < players:GetSize
                        repeat while j < players:GetSize
                        players:Get(i):Get(0) == players:Get(j):Get()



        end

        class Warrior is character

        end

        class Mage is character

        end

        class character
                integer hitPoints
                public action attack(character c)

```

```
                    character:decreaseHitpoint(200)
            end

            public decreaseHitpoint(integer amount)
                    hitPoints = hitPoints - amount
end
```

6.6 Participant #6

```
action main
  integer sum = 0
  Array <Product> prod = basket:Get()
  integer count = 0
  repeat while count<prod:GetSize()
    sum  = sum + prod:GetProd():GetPrise()
  end

  integer count2 = 0

  repeate while count< prod:GetSize()
  if (prod:GetProd == 'oranges' )
    integer numOfOrn = numOfOrn + 1
  else
    integer numOfBan = numOfBan + 1
  end

    integer tripOrn = numOfOrn / 3
    integer discountprice = tripOrn  * 10
    integer normalprice = (numOfOrn - tripOrn) * 15
    integer totalpriceOrn = discountprice + normalprice

    integer fiveBan = numOfBan / 5

  if basket:GetCustomer():IsRegular == true
    discountprice = price * 0.9

end


Task 2

action Main

  Array <Player> team1 = GetTeamPlayers()
```

```
    Array <Player> team2 = GetTeamPlayers()
    team1:Sort()
    team2:Sort()


    integer i = 0
    repeat while i < team2:GetSize()
      if team1:GetPlayer(i):GetPlayerFName() = team1:Get(i+1):GetPlayerFName
         or team1:Get(i):GetPlayerLName() = team1:Get(i+1):GetPlayerLName
      output  `Same team : ' + team1:Get(i):GetPlayerFName  +
         team1:Get(i+1).GetPlayerFName

      else if team1:Get(i):GetPlayerFName() = team2:Get(i):GetPlayerFName or
         team1:Get(i):GetPlayerLName() = team2:Get(i):GetPlayerLName
      output "different teams :" + team1:Get(i):GetPlayerFName  +
         team1:Get(i+1).GetPlayerFName

      else team1:GetPlayer(i):GetPlayerFName() =
         team1:Get(i+1):GetPlayerFName or team1:Get(i):GetPlayerAge() =
         team1:Get(i+1):GetPlayerAge

      output  'Same team : ' + team1:Get(i):GetPlayerFName  +
         team1:Get(i+1).GetPlayerFName + "Same age"
      end
    end
  end

  Task 3

  class StartGame
    action Main

    end
  end

  class Hero
    integer hitpoints = 100
    integer replRate = 10

    action replanishHealth()
    output "Health has been replaneshed from " + hitpoints " to " +
       hitpoints+replRate
    end

    action attack(Hero H)
    end

    action replRes
    end
  end

  class Warrior is Hero
    int fury = 100

    action attack( Hero H)
    integer attackp = hitpoints - 15
    H:hitpoints = attackp
    output H + " has been slayen for " + attackp
    end

    action spattack( Hero H)
    integer attackp = hitpoints - 17
    H:hitpoints = attackp
    integer furyleft = fury - 10
    fury = furyleft
    output H + " has been slayen for " + attackp
    end

    action replRes
    fury = fury+10
    end

  end

  class Mage is Hero
    int mana = 100

    action attack( Hero H)
    integer attackp = hitpoints - 12
    H:hitpoints = attackp
    output H + " has been slayen for " + attackp
    end

    action spattack( Hero H)
    integer attackp = hitpoints - 15
    H:hitpoints = attackp
    integer manaleft = mana - 10
    mana = manaleft
    output H + " has been slayen for " + attackp
    end

    action replRes
    mana = mana+10
    end
  end

  end
```

6.7 Participant #7

```
//Task 1
action Main

  output CalculateTotal(5, 5)
  //test the method
end

action CalculateTotal(integer numberOfOranges, integer numberOfBananas,
    boolean regular) returns number
  integer banana = 4
  integer orange = 5

  number currenTotal = 0

  currenTotal = (numberOfOranges mod 3) * orange + (numberOfOranges/3)*10
  currenTotal = currenTotal + (numberOfBananas mod 5) * banana +
      (numberOfBananas/5)*10

  if regular
    currenTotal = currenTotal*0.9
  end

  return currenTotal
end

//Task 2
use Libraries.Containers.Array
action Main
  Array<player> team1
  Array<player> team2


end

action SameTeamNames(Array<player> team) returns Array<player>
  Array<player> returnArray
  integer i = 0
  integer j = 1

  repeat while i < team:GetSize()
    repeat while j < team:GetSize()
      if team:Get(i):FirstName() = team:Get(j):FirstName() or
         team:Get(i):LastName() = team:Get(j):LastName()
        returnArray.Add(team:Get(i))
        returnArray.Add(team:Get(j))
      end
      j = j+1
    end
    i = i + 1
    j = i+1
  end

  return returnArray
end

action DiffTeamNames(Array<player> team1, Array<player> team2) return
    Array<player>
  Array<player> returnArray

  integer i = 0
  integer j = 0

  repeat while i < team1:GetSize()
    repeat while j < team2:GetSize()
      if team1:Get(i):FirstName() = team2:Get(j):FirstName() or
         team1:Get(i):LastName() = team2:Get(j):LastName()
        returnArray.Add(team1:Get(i))
        returnArray.Add(team2:Get(j))
      end
      j = j+1
    end
    i = i + 1
    j = 0
  end

  return returnArray
end

action SameTeamAge(Array<player> team) returns Array<player>
  Array<player> returnArray

  integer i = 0
  integer j = 1

  repeat while i < team:GetSize()
    repeat while j < team:GetSize()
      if team:Get(i):FirstName() = team:Get(j):FirstName() and
          team:Get(i):Age() = team:Get(j):Age()
        returnArray.Add(team:Get(i))
        returnArray.Add(team:Get(j))
      end
      j = j+1
    end
    i = i + 1
    j = i+1
  end

  return returnArray
end

//Task 3


//Task 4
action Main
  text t = "HenrikGeertsen"

  integer i  = t:GetSize()-1
  text out = ""
  repeat while i > 0
    out = out + t:GetCharacter(i)
    i = i - 1
  end
  output out

  Array<text> a = t:Split("")
  a:Sort()
  i = 0
  out = ""
  repeat while i < a:GetSize()
    out = out + a:Get(i)
    i = i + 1
  end
  output out

  i = 1
  boolean found = false
  integer duplicates = 0
  repeat while i < a:GetSize()
    if (a:Get(i) = a:Get(i-1))
      found = true
    else
      if (found)
        duplicates = duplicates + 1
      end
      found = false
    end
  end
  output duplicates
end
```

6.8 Participant #8

```
use Librarises.Containers.Array

class fruit
  number _price = 0;

  public action RaisePrice(number newPrice)
    me:price = newPrice

class banana is fruit
  number _price = 4

class orange is fruit
  number _price = 5

class bananas
  Array<banana> _bananas
  public action addBanana()
    _bananas:add(banana)

class oranges
  Array<orange> _oranges
  public action addOrange()
    _oranges:add(orange)

class calculator
  public action isRegular(bool isRegular, number price) returns number
    if isRegular = true
      return price = price * 1.10

```

```


action Main
  integer i = 0
  number OTotal = 0
  number BTotal = 0

  oranges orangesLst
  bananes bananasLst
  calculator c

  repeat while not(i = 10)
    orangeLst:addOrange()
    bananasLst:addBanana()

  repeat while i < orangeLst:GetSize()
    Ototal = OTotal + orangeLst:_oranges:Get(i):_price
    if (i mod 3) == 0
      OTotal = OTotal - 5

  regularPrice = c:isRegular(true, OTotal)
  normal = c:isRegular(false, OTotal)

  repeat while i < bananasLst:GetSize()
    Ototal += bananasLst:_bananas:Get(i):_price
    if (i mod 3) == 0
      BTotal = BTotal - 10

  repeat while i



action Main2
  int nrPlayers = 11

  Array<text> fullNames1
  Array<text> fullNames2

  fullNames1:add("martin, fruensgaard, 24")
  fullNames2:add("Tommy, something, 23")

  int i = 0, j = 0;
  repeat while i < fullNames1:GetSize()
    repeat while j < fullNames2:GetSize()
      Array<text> name1 = fullNames1:Get(i):Split(",")
      Array<text> name2 = fullNames2:Get(j):Split(",")

      if(name1:Get(0) = name2:Get(0) or name1:Get(1) = name2:Get(1))
        output "BINGo!<3 2 players: " + fullNames1(i) + " and "
            fullNames2(j)


  int i = 0, j = 0;
  repeat while i < fullNames1:GetSize()
    repeat while j < fullNames2:GetSize()
      Array<text> name1 = fullNames1:Get(i):Split(",")
      Array<text> name2 = fullNames1:Get(j):Split(",")

      if not(name1 = name2)
        if(name1:Get(0) = name2:Get(0) or name1:Get(1) = name2:Get(1))

  int i = 0, j = 0;
  repeat while i < fullNames1:GetSize()
    repeat while j < fullNames2:GetSize()
      Array<text> name1 = fullNames1:Get(i):Split(",")
      Array<text> name2 = fullNames1:Get(j):Split(",")

      if not(name1 = name2)
        if(name1:Get(0) = name2:Get(0) and  name1:Get(2) = name2:Get(2))
```