From Educational Programming to Professional Programming

Group DPT906E15 - Room X.X.XX

4 February - 1 June

Date	Jais Morten Brohus Christiansen
Date	Henrik Vinther Geertsen
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:

Whist on Mobile Devices

Project period:

4 February - 1 June

Project group:

DPT906E15

Participants:

Jais Morten Brohus Christiansen
Henrik Vinther Geertsen
Svetomir Kurtev
Tommy Aagaard Christensen

Supervisor:

Bent Thomsen

Abstract:

Pages: 8

Appendices: 0

Copies: 2

Finished: 1 June 2015

Contents

Part I

Problem Analysis

Chapter 1

Error-Prone Areas for Novices

For a person new to programming, different constructs and concepts can be so confusing that this person might give up without much effort. This area is of great interest to many studies, as it can help future generations in learning programming with ease. But to find solutions, the difficulties that can arise when learning programming and the concepts that follow must be known.

This chapter focuses on the different aspects of learning programming, which can be difficult to grasp for novices. The different aspects and concepts are found by previous studies as well as subjective speculation.

1.1 Syntax and Semantics

As known, a programming language is based on the syntactical rules and the semantic relations. These concepts can be hard to grasp at first, and can be even harder to understand in relation and when used in a practical solution.

One of the most error prone areas for novice programmers is the basic syntax [1]. This consists of brackets, semicolons, commas, and other such symbols, symbolizing control for the program. This problem might relate to an even greater problem in understanding the strict control that is needed when writing code in general. When writing code, even the smallest mistake or forgotten symbol leads to a compiler error. This error margin isn't seen very often in other lines of work, and might discourage the novices from keep trying.

Understanding what a line of code does in itself might be hard for some new programmers. Understanding the connection of the whole program, and what the single line does for the result is even harder. The semantics can lead to confusion, as the program grows bigger. Some times, the novice programmer is even discouraged from even trying, as the connection between the code and what it results in is not clear.

1.2 Programming Paradigms

Different paradigms each have their different difficulties. Many programmers first touch programming through an imperative or procedural approach. Others start out with an object oriented programming language.

Procedural programming has its values in its very straight forward and easily trackable nature. On the other hand, it is hard to see the connection to real world problem solutions, as the very strict text-based structure doesn't resemble these much. Nevertheless, certain tools are used today for teaching, such as Scratch (have we described these yet?), which makes procedural programming a valid learning approach.

Object oriented programming (OOP) has it's values in representing real world problems, and how a solution can be modelled. As OOP is mostly based on classes, the static description of an object, and objects, the dynamic model of a real world phenomenon, the concepts of the paradigm can be easily grasped. Of course, this fact demands a teaching method suitable for the novice programmers being taught. On the other hand, OOP is often in relation to procedural programming seen as not being something else, but the same, only with OO features [1]. This leads to a problem of both understanding the very basic concepts of programming, such as control structures (loops and selections), and understanding the OO approach.

It is discussed widely what approach is the most efficient teaching method (TODO: need some refs here). For instance, some say it is necessary to learn the concepts of OOP before learning to code, and some say the basic constructs are necessary before learning about different paradigms and advanced structures. The procedural approach is being taught in elementary school in different languages (TODO: assumption, need refs). In OOP, the question is often what teaching methods are used to make students understand the concepts of the paradigm. Studies have shown a better effect when teaching about the concepts before actually coding [2], which can be seen as an "object-first approach".

Chapter 2

Text Based Programming Languages

Text based programming languages can be split into two categories; one is the text based educational programming languages for novices, and the other are the general purpose programming languages. Event though general purpose programming languages can be used to teach programming to novices, the two categories are distinguished in where the focus of the design has been. This chapter will explore the programming languages in both of these categories.

2.1 Text Based Educational Programming Languages

One of the first programming languages that added constructs for learning programming was LOGO. It did not have the constructs from the start, but after 12 seventh-grade students[1] worked with LOGO for a year (1968-1969), Seymour Papert, one of the developers of LOGO, proposed the Turtle as a programming domain that could be interesting to people at all ages. He proposed it since the demonstration had confirmed that LOGO was a learnable programming language for novices, but he wanted the demonstration extended to lower grades, ultimately preschool children. Constructs for Turtles was then added to LOGO and has since been widely adopted in other programming languages such as SmallTalk and Pascal, and more recently Scratch.

A Turtle can be a visual element on a screen or a physical robot. In Scratch, the Turtle can be any sprite chosen by the user. In LOGO the Turtle is controlled by a set of commands which are:

- FOWARD X, moves the Turtle X number of Turtle steps in a straight line
- RIGHT X, turns the Turtle X number of degrees in a clockwise direction
- LEFT X, turns the Turtle X number of degrees in a counter clockwise direction
- PENDOWN, makes the Turtle draw

[1] From Muzzy Junior High School in Lexington, Massachuseets.

- PENUP, makes the Turtle stop drawing

These commands make up the essence of Turtle programming and is also present in the other languages which has implemented Turtle programming. Some languages has expanded on these commands e.g. in Scratch, one can change the color, size and shade of the pen.

TODO: What is turtle programming supposed to teach?
TODO: Other languages
TODO: GRAIL
TODO: transition to visual programming

2.2 General Purpose Programming Languages

TODO: Why is it the goal to learn these languages?
TODO: Why can't we stick with the educational ones?
TODO: Why don't we begin with these?
TODO: Tools for learning e.g. BlueJ

Part II

Bibliography

Bibliography

[1] A. R. Sandy Garner, Patricia Haden, "My program is correct but it doesn't run: A preliminary investigation of novice programmers' problems," *ACE '05 Proceedings of the 7th Australasian conference on Computing*, vol. 42, 2005. 1.1, 1.2

[2] S. Xinogalos, "Object-oriented design and programming: An investigation of novices' conceptions on objects and classes," *ACM Transactions on Computing Education*, vol. 15, no. 3, 2015. 1.2