GymInf | Informatikausbildung für Gymnasiallehrkräfte | Formation en informatique destinée aux enseignant-e-s au gymnase

Revealing Programming Language Abstractions

An Excerpt of nand-to-tetris – in Reverse – Using Smalltalk

GymInf Individual Project

Simon Bünzli
from
Bern, Switzerland

Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

01. August 2025

Prof. Dr. Timo Kehrer, Prof. Dr. Oscar Nierstrasz

Software Engineering Group
Institut für Informatik und angewandte Mathematik
University of Bern, Switzerland

UNIVERSITÄT BERN

UNIVERSITÉ DE NEUCHÂTEL

UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

Abstract

Not an *abstract* yet, but the original project description:

Ziel des Projekts ist, ein empirisch abgestütztes Instrument für den Programmier-Unterricht am Gymnasium zu entwickeln, in welchem Schüler:innen verschiedene Abstraktionsebenen interaktiv erleben können.

Auf der Basis von Processing mit Python Syntax (https://py.processing.org/) soll der einerseits der visuelle Ablauf eines Programms, aber auch die Parsing-Schritte und die Übersetzung in Byte-Code Seite-an-Seite sicht- und untersuchbar gemacht werden, damit Schüler:innen die Auswirkungen ihres Programmcodes auf die Maschine live erleben können.

Die Entwicklung des Produkts wird theoretisch begleitet und das Produkt selbst empirisch geprüft werden.

Als Basis der Umsetzung dient Glamorous Toolkit, eine Entwicklungsumgebung basierend auf Smalltalk/Pharo, welche u. a. von Oscar Nierstrasz für Master- und Doktoratsstudiengänge weiterentwickelt worden ist.

Contents

1 Introduction

Programming with Processing vs. ”Little Man Computer” or ”Human Resource Machine”

2

Leaky Abstractions when Teaching Programming

Before introducing the product of this thesis in chapter 4, we first introduce the problem it should help solve in this chapter: How abstractions involved in programming are taught.

In detail, we first introduce the concept of (leaky) abstractions in multitier architectures in section 2.1; show how these are discussed in didactic literature in sections 2.2; and how common IDEs already support handling these difficulties (in section 2.3).

2.1 Multitier Architectures and (Leaky) Abstractions

In order to handle complexities arising in both theoretical and practical computer science **¡citation!**, subjects are split into multiple layers to be described, investigated and used separatedly.

Common such multitier architectures taught at high school level are the networking stack (either the seven layered ISO architecture or the simplified four layered DoD architecture **¡citation!**) or the software-hardware stack ranging from apps and hardware abstracting OS down to transistors consisting of e. g. silicium atoms.

¡diagram of such an architecture!

Ideally, in such architectures all layers above the layer to be investigated can be ignored (beyond what the layer will be used for) and all the layers below can be abstracted away into a nicely defined interface.

As such, programming should be possible to be done independent of hardware and even the operating system, in the same way that natural languages can be taught independently of body or mind of the students.

In his article ”The Law of Leaky Abstractions” [13] introduces the concept of *leaky abstractions*, claiming that for all non-trivial such architectures, details of lower layers are to some degree bound to bleed through to upper layers. In other words, in practice complex interfaces tend to be incomplete or ’leaky’.

In teaching computer science, such leaky abstractions occur repeatedly, e. g. when an app doesn’t run on a different device (with either the OS or the processor architecture leaking); or when a document seemingly can’t be saved (with either the file system or different kinds of apps leaking).

2.2 Didactic Approaches

See e. g. [12], [8], [6] or [7] only focusing on one aspect

2.2.1 Teaching Top Down

Working downwards from gaming, as in [16]

2.2.2 Teaching Bottom Up

Running Tetris on NANDs as described in [15], [10]

2.3 Abstractions in IDEs

Integrated development environments used for programming offer a variety of different views on a program beyond its source code and its runtime output. The popular Visual Studio Code offers e.g. through extensions step-by-step debugging with variables and the call stack listed [3]. This is mirrored in most other full fledged IDEs such as PyCharm [1] or Eclipse [2].

And while such IDEs through appropriate extensions even allow inspecting Python bytecode, the respective views are usually overwhelming for programming novices and thus rather targetted at professional developers than high school students.

As a remedy, several teaching oriented IDEs have been developped, such as ”Code with Mu” which offers a minimal command set and still allows runtime inspection [14]; or Thonny which had the goal to visualize runtime concepts beyond what IDEs offered at the time [4, p. 119]:

On the one hand, Thonny shows intermediary steps during expression evaluation. This demonstrates that statements are not evaluated in one go, but indeed in a predetermined order operation by operation.[1]

[1] In professional IDEs, intermediary results are usually available by hovering over a specific operator with the order of evaluation being left to the user to determine.

3 Technical Background

Background knowledge required for understanding the following chapters.

3.1 Processing

Brief overview over the ”Processing” programming language (along [11]) and reasons for using it.

3.2 Glamorous Toolkit

Brief introduction into GT for the uninitiated and reasons for using it. ([5])

3.3 Moldable Development

Referring to [9].

4
Processing Abstractions

4.1 Development of ”Processing Abstractions”

Excerpts from gt-exploration Lepiter pages

4.2 Abstraction Levels

For each a short problem description and a presentation of the chosen approach:

4.2.1 Source Code

4.2.2 Abstract Syntax Tree

4.2.3 Transpilation/IR

4.2.4 Machine Code

4.2.5 Output

5 Teaching with PA

How to use it

5.1 Lesson on Computer Architecture

Using PA to demonstrate what happens under the hood when running a program in a high level language.

5.2 Lesson on Compilers

Using PA to demonstrated the steps of lexing, parsing, transpiling, compiling and optimizing.

5.3 Lesson on Introduction to Programming

Using PA as a live programming environment.

6 PA in Practice

PA has been used twice with students (on 2025-05-12 and 2025-06-30).

6.1 First Round

6.1.1 Setting

6.1.2 Observations

6.1.3 Student Feedback

6.1.4 Learnings

6.2 Second Round

6.2.1 Setting

6.2.2 Observations

6.2.3 Student Feedback

6.2.4 Learnings

7 Conclusion

7.1 Future Work

A

Installing and Using `Processing` Abstractions

B

Data from Questionnaires

Bibliography

[1] Debug your python code with pycharm.

[2] Pydev - python ide for eclipse.

[3] Python debugging in vs code.

[4] Aivar Annamaa. Introducing thonny, a python ide for learning programming. In *Proceedings of the 15th Koli Calling Conference on Computing Education Research*, Koli Calling '15, pages 117–121, New York, NY, USA, 2015. Association for Computing Machinery.

[5] Tudor Gîrba and Oscar Nierstrasz et al. *Glamorous Toolkit*. feenk.

[6] Stefan Hartinger. *Programmieren in Python*. Universität Regensburg, 2020.

[7] Irene Lee, Shuchi Grover, Fred Martin, Sarita Pillai, and Joyce Malyn-Smith. Computational thinking from a disciplinary perspective: Integrating computational thinking in k-12 science, technology, engineering, and mathematics education. *Journal of science education and technology*, 29(1):1–8, 2020.

[8] Eckart Modrow and Kerstin Strecker. *Didaktik der Informatik*. De Gruyter Studium. De Gruyter Oldenbourg, München, 2016.

[9] Oscar Nierstrasz and Tudor Gîrba. Moldable development patterns. 2024.

[10] Noam Nisan and Shimon Schocken. *The elements of computing systems : building a modern computer from first principles*. The MIT Press, Cambridge, Massachusetts, second edition edition, 2021 - 2021.

[11] Casey Reas and Ben Fry. *Processing: A Programming Handbook for Visual Designers and Artists*. The MIT Press, 2 edition, 2014.

[12] Sigrid Schubert and Andreas Schwill. *Didaktik der Informatik*. Spektrum Akademischer Verlag, Heidelberg, 2. auflage edition, 2011.

[13] Joel Spolsky. The law of leaky abstractions, 2002.

[14] Nicholas H. Tollervey. The visual debugger, 2023.

[15] Ünal Çakıroğlu and Mücahit Öztürk. Flipped classroom with problem based activities: Exploring self-regulated learning in a programming language course. *Journal of Educational Technology & Society*, 20(1):337–349, 2017.

[16] David Weintrop and Uri Wilensky. Playing by programming: Making gameplay a programming activity. *Educational Technology*, 56(3):36–41, 2016.