
This is an electronic reprint of the original article.
This reprint may differ from the original in pagination and typographic detail.

Lukkarinen, Aleksi

Supporting Media Computation in Programming Education:

Published: 17/06/2016

Document Version

Publisher's PDF, also known as Version of record

Please cite the original version:

Lukkarinen, A. (2016). *Supporting Media Computation in Programming Education: A Class Library for Bitmap Processing Using Scala*. [Master's thesis, Department of Computer Science, School of Science, Aalto University]. Aalto University. <http://urn.fi/URN:NBN:fi:aalto-201606172549>

This material is protected by copyright and other intellectual property rights, and duplication or sale of all or part of any of the repository collections is not permitted, except that material may be duplicated by you for your research use or educational purposes in electronic or print form. You must obtain permission for any other use. Electronic or print copies may not be offered, whether for sale or otherwise to anyone who is not an authorised user.

Aalto University
School of Science
Degree Programme in Computer Science and Engineering

Aleksi Lukkarinen

Supporting Media Computation in Programming Education

A Class Library for Bitmap Processing Using Scala

Master's Thesis

Espoo, May 23, 2016

Supervisor:	Professor Lauri Malmi
Advisors:	Juha Sorva, D.Sc. (Tech.) Otto Seppälä, D.Sc. (Tech.)

Author Aleksi Lukkarinen

Title of Thesis Supporting Media Computation in Programming Education:
A Class Library for Bitmap Processing Using Scala

Degree Programme Computer Science and Engineering

Thesis Supervisor Professor Lauri Malmi**Code of Professorship** T-106

Department Computer Science and Engineering

Thesis Advisors Juha Sorva, D.Sc. (Tech.); Otto Seppälä, D.Sc. (Tech.)

Date May 23, 2016 **Number of Pages** xx + 106 + 3 appendices **Language** English

Abstract

The objective of the Thesis was to identify opportunities for applying Media Computation in four courses that teach introductory computer science and programming with the Scala programming language and are aimed at computer science majors at Aalto. Media Computation is an approach for contextualizing computer science and programming education to make it appear more interesting and motivating for students.

The main components of this thesis are (1) a classification of the currently available teaching contextualization methods for programming and computer science, with a focus on Media Computation; and (2) a research prototype which supports students by offering cognitive scaffolding and improves the presentation of bitmap-based media on the interactive Scala shell of Scala IDE.

The Thesis examines the foundation of contextualization and Media Computation in theories of motivation. This thesis formulates a classification that provides a coarse-grained overall picture of tool-based contextualization methods and a more fine-grained description of ways to integrate media inside integrated development environments, a form of Media Computation. In addition, on the basis of several textbooks that apply Media Computation, the Thesis presents a classification of current genres of Media Computation. Moreover, it reviews software applications that have already been used, or would be suitable to be used, for Media Computation.

The research prototype consists of two Scala-based class libraries and a bespoke version of the Eclipse-based Scala IDE. The first of the class libraries, SMCL (Scala Media Computation Library), enables the actual bitmap processing. The latter one, in turn, offers an interface for the interactive Scala shell both to query SMCL for and to display metadata, such as bitmaps, that is related to the execution results of code fragments. SMCL enables both pixel- and compositing-based Media Computation with RGBA-formatted (red, green, blue, alpha) bitmaps. It also contains a simple bitmap viewer for cases in which the development environment cannot display bitmaps or does not support SMCL.

The requirements set for the prototype were almost fully realized, and a small-scale pilot testing proved the functionality offered by the prototype to be useful. With further development of the prototype it is most likely possible to produce a tool, with which programming education can be improved to be more diverse, interesting, and motivating.

Keywords Scala, Media Computation, contextualized teaching, introductory programming education, motivation, bitmap, graphics, class library



Tekijä Aleksi Lukkarinen

Työn nimi Medialaskentaa ohjelmointiopetukseen:
Scala-pohjainen kirjasto bittikarttojen käsittelyyn

Koulutusohjelma Tietotekniikka

Valvoja Professori Lauri Malmi

Professuurikoodi T-106

Laitos Tietotekniikka

Työn ohjaajat Juha Sorva, TkT; Otto Seppälä, TkT

Päiväys 23. toukokuuta 2016

Sivumäärä xx + 106 + 3 liitettä

Kieli Englanti

Tiivistelmä

Tämän diplomityön tavoitteena oli tutkia mahdollisuuksia medialaskennan soveltamiseen neljällä Aalto-yliopiston Tietotekniikan laitoksen pääaineopiskelijoille suunnatulla kurssilla, jotka opettavat tietojenkäsittelytieteen ja ohjelmoinnin alkeita Scala-ohjelmointikielellä. Medialaskenta on näiden aiheiden opetukseen suunnattu viitekehyksellistämismenetelmä, joka tähtää opetuksen muokkaamiseen opiskelijoiden näkökulmasta mielenkiintoisemmaksi ja motivoivammaksi.

Työn tärkeimmät osat ovat (1) tietojenkäsittelytieteen ja ohjelmoinnin opetukseen suunnattuja viitekehyksellistämismenetelmiä käsittelevä ja medialaskentaan painottuva luokittelu sekä (2) tutkimusprototyyppi, joka tarjoaa opiskelijoille kognitiivista tukea sekä parantaa bittikarttapohjaisen median esittämistä Scala IDE -kehitysympäristön Scala-komentorivillä.

Työssä haetaan perusteita sekä viitekehyksellistämisen että medialaskennan toimivuudelle motivaatioteorioista. Kokonaiskuvan työkalupohjaisista viitekehyksellistämismenetelmistä saa työn tarjoamasta karkeasta luokittelusta; medialaskennan osalta tämä luokittelu tarkentuu tapoihin esittää mediaa integroiduissa kehitysympäristöihiss. Lisäksi työ esittelee medialaskentaa soveltaviin oppikirjoihin pohjautuvan luokittelun medialaskennan suuntauksista sekä tutustuttaa joihinkin medialaskennassa jo käytettyihin tai siihen soveltuviin ohjelmistoihin.

Tutkimusprototyyppi koostuu kahdesta Scala-kielisestä luokkakirjastosta sekä Eclipse-pohjaisen Scala IDE -kehitysympäristön mukautetusta versiosta. Luokkakirjastoista ensimmäinen, SMCL (Scala Media Computation Library), mahdollistaa varsinaisen bittikarttojen käsittelyn, kun taas jälkimmäinen tarjoaa rajapinnan Scala-komentorivillä suoritettujen ohjelmakoodikokonaisuuksien suoritustuloksia koskevan metatiedon, kuten bittikarttojen, siirtämiseen SMCL:ltä Scala-komentorivillä esitettäväksi. SMCL mahdollistaa medialaskennan rgba-muotoisilla (red, green, blue, alpha) bittikartoilla sekä kuvapistepohjaisesti että bittikartoista kokonaisuuksia koostaen. SMCL sisältää myös yksinkertaisen ohjelman bittikarttojen tarkastelemiseksi tapauksissa, joissa kehitysympäristö joko ei kykene esittämään bittikarttoja tai ei tue SMCL:ää.

Prototyypille asetetut vaatimukset toteutuivat lähes kokonaan ja pienimuotoinen pilottitestaus osoitti prototyypin tarjoaman toiminnallisuuden hyödylliseksi. Prototyyppiä jatkokehittämällä voidaan todennäköisesti tuottaa työkalu, jonka avulla ohjelmointiopetusta voidaan kehittää entistä monipuolisemmaksi, mielenkiintoisemmaksi ja motivoivammaksi.

Asiasanat Scala, medialaskenta, viitekehyksellistetty opetus, ohjelmoinnin alkeisopetus, motivaatio, bittikartta, grafiikka, luokkakirjasto

Contents

List of Figures	vii
List of Tables	ix
List of Code Examples	xi
Preface	xiii
Abbreviations and Definitions	xv
1 Introduction	1
1.1 Computers and Programming in Today's Society	1
1.2 Introductory Progr. Courses at Aalto University's Dept. of CS	4
1.3 Research Questions, Goals, and Process	6
1.4 Scope Restrictions	8
1.5 Existing Literature	9
1.6 Computing Education Research	10
1.7 Classifications and Relations to Research Disciplines	11
1.8 Content and Notations	13
2 Motivation and Contextualization in Education	15
2.1 Motivation	15
2.2 Contextualized Content	19
2.3 Apch. for Contextualization and Multimedia Usage	21
2.3.1 Solutions Essentially Combining Software and Hardware	22
2.3.2 Essentially Software-Based Solutions	23
2.3.3 Essentially Hardware-Based Solutions	26
3 Media Programming as a Context	27
3.1 Three Views into Media	27
3.2 Defining Media Programming	28
3.3 Effects of Media Programming	32
3.4 Applying Media Programming	33
3.5 Tools Enabling Media Programming	35
4 Prototype: Media Computation in Scala	47
4.1 Stakeholders	47
4.2 Requirements	48
4.3 Professional IDEs for Scala	49
4.4 Structure	51
4.5 Functionality	51
4.5.1 Scala IDE's REPL	51

4.5.2	SMCL	52
4.6	Exercise Examples	56
4.6.1	Expressions and the Little Quilt Language	56
4.6.2	Repetition Structures and Pixel Iteration	58
5	Prototype Implementation	63
5.1	Scala Media Computation Library	63
5.1.1	Technologies and External Dependencies	63
5.1.2	Packages	64
5.1.3	Settings	65
5.1.4	Colors	66
5.1.5	Bitmap Processing	66
5.2	Customized Scala IDE and Metadata Query	68
5.3	The Research Prototype as a Project	71
6	Evaluation	75
6.1	Requirements for the Research Prototype	75
6.2	Pilot Study: The Prototype’s Applicability	78
6.3	Overall Results	79
7	Conclusion	83
	Bibliography	85
Appendices		
A Literature Data Providers		
B Code Examples for IDEs Supporting Media Programming		
C Selected Resources by Topic		

List of Figures

i	Badges in Audible for Windows 10	xv
1.1	Introductory programming courses of Aalto University’s Dept. of CS	5
1.2	Relations between this thesis and computing, education and psychology . .	12
1.3	Relations between this thesis and computing education research areas . . .	13
1.4	Bitmaps used as source images	14
2.1	Classification of tool-based CS contextualization approaches	24
3.1	Jython Environment for Students: Bitmap manipulation	36
3.2	Kojo: A recursive tree	37
3.3	Snap!: Drawing turtle graphics and playing sounds	38
3.4	Pixly: Bitmap manipulation	39
3.5	EarSketch: Clip-based sound waveform compositing	40
3.6	MATLAB: Bitmap manipulation	41
3.7	Maple: Bitmap manipulation	42
3.8	DrRacket: Bitmap manipulation	43
3.9	Mathematica: Bitmap manipulation and integrated multimedia	45
4.1	Prototype: The most important stakeholders	47
4.2	IntelliJ IDEA with Scala plugin	50
4.3	Prototype: The main components	51
4.4	Prototype: Customized Scala IDE REPL started	52
4.5	Prototype: Representation of colors	54
4.6	Prototype: SMCL’s bitmap viewer	55
4.7	Exercises: Quilts created with the Little Quilt Language	57
4.8	Exercises: Results of pixel iteration examples	58
5.1	SMCL: External dependencies	64
5.2	SMCL: Packages and their most essential mutual dependencies	65
5.3	SMCL: Settings-related core classes and objects	65
5.4	SMCL: Color-related core classes and traits	66
5.5	SMCL: Bitmap-related core classes, traits, and types	67
5.6	SMCL: Reduction of image quality	68
5.7	SMCL: Classes, traits, and objects of the external interfaces	69
5.8	Prototype: Project web page at GitHub	72
6.1	The web page distributing testing resources for the prototype	79

List of Tables

- 1.1 Research questions and goals 7
- 1.2 Typographical notations 14
- 2.1 Examples of software development environment classification criteria 21
- 3.1 The current genres and exercise types of Media Programming 31
- 4.1 Functional requirements for the research prototype 49
- 6.1 Replicating bitmap operations from *Picturing Programs* 76
- 6.2 Replicating bitmap operations from *How to Design Programs* 77
- A.1 The data providers utilized to obtain literature A-1

List of Code Examples

The Scala Media Computation Library

4.1	The global settings and their default values	53
4.2	Creating quilts	57
4.3	Pixel iteration using <i>for loop</i>	59
4.4	Pixel iteration using <i>for comprehension</i>	59
4.5	Pixel iteration using <i>while loops</i>	60
4.6	Pixel iteration using <i>foreach</i> method and <i>lambda function</i>	60
4.7	Pixel iteration using <i>pixel iterator</i> and <i>while loop</i>	61

IDEs Supporting Media Programming

B.1	Jython Environment for Students: Bitmap manipulation	B-1
B.2	Kojo: Turtle graphics	B-2
B.3	Snap!: Turtle graphics and sound playing	B-3
B.4	Pixly: Bitmap manipulation	B-4
B.5	Tunely: Sound manipulation	B-4
B.6	EarSketch: Clip-based sound waveform compositing	B-5
B.7	MATLAB: Bitmap manipulation 1	B-6
B.8	MATLAB: Bitmap manipulation 2	B-7
B.9	Maple: Bitmap manipulation	B-8
B.10	DrRacket: Bitmap manipulation	B-9
B.11	Mathematica: Multimedia integrated with program code	B-10
B.12	Mathematica: Image manipulation	B-11

Preface

This Master’s Thesis was carried out for the Learning + Technology Research Group at the Department of Computer Science of Aalto University. First and foremost, I express my gratitude to professor Lauri Malmi for supervising this Thesis and giving me patient instruction. Furthermore, I thank both my advisors, university lecturer Juha Sorva and university teacher Otto Seppälä, for constructive and apt feedback, as well as for helping with many practicalities of this work. I also thank Lauri, Juha, Otto, and Aalto University for the possibility to work at the Department during the writing of this Thesis. Juha suggested the topic of Media Computation combined with Scala in the first place.

Moreover, my compliments go for the people with whom I have shared an office at the Department—Teemu Sirkiä, Lassi Haaranen, and Teemu Lehtinen—for being happy and encouraging. Finally, I thank my sister and her children, my brother, and my parents.

Espoo, May 23, 2016

Aleksi Lukkarinen

Bachelor of Engineering, Student of Software Technology
Degree Programme in Computer Science and Engineering
School of Science
Aalto University

This Thesis was typeset with X_gLaTeX and TeX Live 2015, using Latin Modern and Lato as the primary type families for body text. TeXstudio 2.10 was used as an editor. Figures other than screenshots were created with Microsoft Visio 2016. A private GitHub repository was used to contain all source resources.

Abbreviations and Definitions

achievement-goal theory

A theory of motivation, which encapsulates an idea of person's attitude towards a task (see § 2.1 on page 18).

ACM

Association for Computing Machinery. An international society for computing professionals, researchers, and educators. One of its purposes is to share information, and as one method to accomplish that it publishes and archives computing-related magazine and conference articles.

API

See *Application Programming Interface*.

Application Programming Interface

A set of programmatic resources, such as constants, data structures, sub-programs, and classes, which a software component offers for other software components to enable mutual communication.

attribution theory

A theory of motivation, which models person's desire to understand some phenomenon (see § 2.1 on page 16).

badge

In the context of *gamification*, refers to an image received as a prize for a certain behavior. These images are being utilized in computer software from games to more serious applications. As an example, Figure i below exhibits some of the badges the users of the Audible software application are able to receive by diverse usage of the application's features.

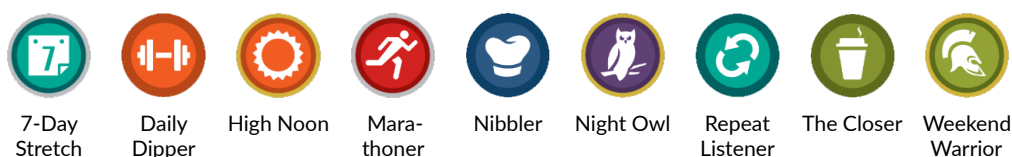


Figure i. Some of the badges in Audible for Windows 10 version 10.3.6.0.

The forms of badges being utilized in some educational institutions include stamps, stickers, and victorian die-cuts as well as medals and trophies.

behaviorism

A theory of learning, which concentrates only on learner's observable behavior and its changes when learning takes place (see § 2.1 on page 16).

blended learning

Several different learning environments are “blended” together. Often, parts of the teaching material is delivered in an electronic format, so that both traditional classroom teaching and computer-based self-studying are blended together.

BYOD *Bring Your Own Device.* A practice according to which students can use their own portable devices in schools (and workers in workplaces) to do their work [e.g., MCM15; JOH14].

CER See *Computing Education Research*.

cognitive load

The amount of work a learner has to do in their working memory during a learning process.

cognitive scaffolding

Planned support that is given to a student so that they can work with material that as a whole is above the level of their current understanding. The scaffolding can be provided for instance by teachers, peer students, and learning materials and environments.

Computing Education Research

An interdisciplinary field of scientific research concerning education in computing sciences (see § 1.6).

contextualization

The process of organizing teaching in such a way that the concepts that are intended to be learned are presented from the viewpoint of a different topic—the *context*—that is intended to be interesting, meaningful, and useful for the learners (see Chapter 2). For example, image processing, games, and robotics are examples of such contexts.

DMX *Digital MultipleX.* A standardized data transfer format to be used for controlling lighting and other stage equipment (e.g., smoke machines and hazers) with both lighting control consoles and general-purpose computers [PLA15].

embedded system

A computer—with a case-specific controlling software—inside of another device not appearing to be a regular general-purpose computer. Examples include toys, thermometers, cell phones, calculators, digital cameras, media players, televisions, computer peripherals (like keyboards, mice, displays, printers, and mass storage), vehicle control and diagnostics, traffic control, and power distribution grid substations as well as industrial instrumentation and automation. These days one does not buy a washing machine but a computer capable of and programmed to wash laundry by driving motors and monitoring parameters measured by several kinds of sensors.

ECTS *European Credit Transfer and Accumulation System.* A system to standardize grading and workload calculation of courses in higher education to enable Europe-wide comparison and transfer of academic achievements.

- ESKO** *Elektroninen SarjaKOmputaattori*. The first computer constructed in Finland, although the design was partly acquired from the Germany. It utilized vacuum tubes, ferrite rings for memory, and punched tape for entering instructions and data.
- expectancy-value theory**
A theory of motivation, which tries to approximate reasons for *why* people would be motivated and why they would choose one task over another (see § 2.1 on page 17).
- flipped classroom**
A teaching approach, in which students familiarize themselves with new concepts mostly themselves before lectures, and the classroom time is used for learning in interaction with teacher and student peers [e.g., MAR14B; VUK14; BS15; WAT15C]. This contrasts with the traditional approach, in which teacher introduces the new concepts in class and students practice them doing homework assignments by themselves.
- GNU** A recursive acronym that stands for *GNU's Not Unix* and refers to a project that develops a free operating system. Today, many people know this operating system as Linux, despite that the name Linux actually refers to its kernel (which was originally developed by Linus Torvalds in 1991) only and that a proper name for the whole distribution would be *GNU/Linux*—the GNU operating system with Linux being used as its kernel. The kernel can be changed, but the rest of the operating system still is the GNU. [FRE14.]
- gamification**
The process of introducing game-like elements that encourage students to study more. Receiving achievement *badges* (see above) is one simple form of gamification.
- GUI** *Graphical User Interface*. A type of user interface that facilitates graphical elements as the means of interaction.
- HTML** *Hypertext Markup Language*. A language that is used to describe the basic structure of web page content.
- individual interest**
An interest type that is caused by intrinsic motivation (see § 2.1 on page 18).
- IDE** *Integrated Development Environment*. The most basic tool in traditional programming is a text editor that is extended to support a more or less rich set of functionality needed in program code handling and processing. By integrating external development tools, like interpreters, preprocessors, compilers, linkers, build tools, packaging and publishing utilities, database administration and querying, testing utilities, version control systems, server management, and so on, the programmer's text editor starts to transform into a full-fledged integrated development environment (see § 2.3.2 and § 3.5).

IEEE	<i>Institute of Electrical and Electronics Engineers.</i> An international technical professional organization devoted to further technical innovation. Among other things, it publishes and archives (computing-related) standards as well as magazine and conference articles.
Java	A primarily object-oriented programming language, into which some characteristics suitable for functional programming have been introduced in its latest revisions. Programs developed in Java are compiled into so called bytecode that is executed by virtual machines called collectively as the Java Virtual Machine (JVM).
Java Development Kit	In this thesis refers to the official software distribution that contains the basic tools (such as compiler) that are needed to develop Java-based programs. There are also other implementations of the JDK tools more or less similar to the ones that the official JDK distribution contains.
Java Virtual Machine	A virtual machine that is developed to execute programs expressed in the Java’s “machine language” called bytecode. There exist different kinds of JVMs with their unique characteristics.
JDK	See <i>Java Development Kit</i> .
JVM	See <i>Java Virtual Machine</i> .
Maslow’s hierarchy of needs	A theory that uses five levels to describe the basic needs of human beings (see § 2.1 on page 16).
Media Computation	A teaching contextualization method, which was developed at the Georgia Institute of Technology at the beginning of the 2000s (see Chapter 3).
Media Programming	A subset of Media Computation, as defined in the § 3.2.
MIDI	<i>Musical Instrument Digital Interface.</i> A standard for transferring information (primarily) between music-related computers, including musical instruments, sound modules, effects, sequencers, and recorders [MID15]. In addition to event- and note-based information, there are so-called system exclusive messages that can be used to transfer practically anything (such as settings, sound waveforms, and screen images), as long as both transmitting and receiving computers agree on the data format used in those messages.
Massive Open Online Course	An umbrella term for several types of courses, the materials of whose are accessible over network for (practically) anyone (cf. SPOC below).
MOOC	See <i>Massive Open Online Course</i> .

PDA *Personal Digital Assistant.* A computer that resembles modern mobile phones, the capabilities of which typically include at least maintaining calendar and notes, including giving alarms. Many of them have other applications, such as calculators, world clocks, personal bookkeeping, dictionaries, and games. As the modern mobile phones offer all of those features and much more, they have taken the place of separate PDAs as the most basic kind of portable computer people carry with them all the time.

read-evaluate-print loop

This term has several interpretations, but in this thesis it refers to programming language compiler’s/interpreter’s and programmer’s interaction that has the following characteristics:

- Single and possibly independent pairs of code fragments and their execution results are displayed as a browsable (and possibly editable) list that contains the (full) history of any code fragments executed during a single REPL session.
- New pairs of code fragments and their execution results are appended to the end of that list by executing new code fragments.
- Execution context is preserved between executions. All the code fragments of a single REPL session share the same execution context.
- Well-constructed REPLs allow execution of existing code fragments independently from the other list content (although in the same single execution context), after which the execution results associated with the executed code fragments are updated.

There are REPL implementations for many programming languages—even Java is finally getting an official REPL implementation called Java Shell in the upcoming Java 9 release [e.g., BUR16; ORA16]. Furthermore, Mathematica’s notebooks, for instance, can be used effectively as both REPL and *worksheet* (see below).

REPL See *read-evaluate-print loop*.

Scala A programming language combining object-oriented and functional programming paradigms. It was originally designed by Martin Odersky, and currently it has a compiler to target the Java Virtual Machine.

Scala Media Computation Library

A part of the research prototype constructed for this thesis (see Chapter 4 onwards).

self-efficacy theory

A theory of motivation, which is concerned with individual’s beliefs about their own possibilities to finish an undertaking (see § 2.1 on page 17).

self-determination theory

A theory of motivation, which classifies individual’s motivation with a six-step scale (see § 2.1 on page 17).

situational interest

A type of interest that is temporally short-lived and supported by prevailing context (see § 2.1 on page 18).

Small Private Online Course

A course, the materials of which are accessible over network for only a small group of participants, such as the students participating in a single realization of a university course (cf. MOOC above).

SMCL See *Scala Media Computation Library*.

SPOC See *Small Private Online Course*.

STEM *Science, Technology, Engineering, Mathematics*.

STEAM *Science, Technology, Engineering, Art, Mathematics*.

UI *User Interface*. The means of interacting with something, such as computers or computer software.

UML *Unified Modeling Language*. A graphical language for modeling computational systems, the requirements set for them, as well as the interaction between them and their environment.

USB *Universal Serial Bus*. A standard for transferring data between computers over so-called USB cables.

worksheet In the context of editing and executing program code, this term has several interpretations. In this thesis, however, it refers to a traditional text editor with the following extension: When the program contained by it is executed, the execution results—usually only those of the top-level statements—are inserted into the program code as comments. The execution context is not preserved between two executions of the editor content as a whole. As a curiosity, Wolfram Mathematica’s notebooks can be used effectively as both worksheet and REPL (see above).

XML *Extensible Markup Language*. A standard for transferring structured information between computer programs.

Chapter 1

Introduction

This thesis is concerned with *Media Programming*, which is later defined (§ 3.2) as a hyponym of *Media Computation*—a contextualization approach for introductory programming teaching. Contributions of this thesis are as follows:

- A classification of tool-based contextualization approaches (§ 2.3). It is coarse-grained in general but focuses on tools related to Media Programming.
- A classification of the main genres and exercise types of Media Programming, based on Guzdial’s and Ericson’s Media Computation curriculum (§ 3.4).
- A prototype that aims to enable Scala-based Media Programming (Chs. 4 and 5).
- From the implementation phase of the prototype, observations concerning problems in implementing functionality related to Media Programming for Scala-based introductory programming courses at the Department of Computer Science of Aalto University.

This first chapter introduces the reader to the thesis. The chapter begins by taking a glance at the position of computers and programming in today’s society (§ 1.1) as well as the current introductory programming courses at the Department of Computer Science of Aalto University, which is one of Finland’s institutions that educate the engineers of the future (§ 1.2). Then the reader is presented with the research setting of this thesis (§ 1.3), a few general restrictions to its scope (§ 1.4), as well as a quick overview of the existing literature related to this thesis (§ 1.5). They are followed by a short description of computing education research (CER) as a research discipline (§ 1.6), and a review of this thesis in relation to it (§ 1.7). As a conclusion, an overview of the following chapters as well as some notation-related aspects are presented (§ 1.8).

1.1 Computers and Programming in Today’s Society

Computers and computer-based automation play a very significant role in a modern society. After the concept of the microprocessor was born at the end of 1960s, general-purpose computers, let alone embedded systems, have gradually become everyday items. Meanwhile, our society has been changing fundamentally: People become more and more dependent upon information technology, as more and more administrative processes are being computerized and made accessible via the Internet. Even many senior citizens have had to learn to use some browser-based services to take care of their daily affairs.

Computerization and continuous diminution of humans performing repetitive tasks may be strange for the older age groups. Children, on the other hand, are growing up in a society, in which computers as well as computer-driven systems and processes are taken for granted. Even children who are not yet in school learn to use computing devices, such as desktop computers, tablets, handheld devices, terminals, and information screens, as well as Internet-based services, from the end-user’s perspective based on simple cognitive schemata they construct based on their experiences: A one-and-a-half-year-old child, such as my nephew, who speaks but a few words, already understands how to browse photos using a gesture-driven user interface of a cell phone.

However, considering that computers and computer-driven information systems have become such an integral and quotidian part of our society, there has been some important shortcomings involved. Firstly, while the people use general-purpose computers on a daily basis, they do not necessarily perceive them as what they really are—ordinary electrical circuits designed by engineers (that is, ordinary people) to be programmable to perform specific tasks. Furthermore, while there are more and more embedded systems around us, the Everyman does not recognize or consider them as such. They do not understand that all of them are essentially computers and logically behave like any general-purpose computer, even if they might not have exactly the same appearance or the same kind of interfaces as their general-purpose cousins.

Even more importantly, the man on the street currently does not understand the concept of programming and is not able to create even simple programs at a conceptual level himself. The ability to understand both data processing algorithms and computers’ structure and operation as well as to develop computer software, has traditionally been limited to the relatively few people, who have taught themselves as a hobby or have completed some studies in computer science (CS), perhaps for a professional career in software development industry.

To rectify the situation, it is important to ensure that future generations can properly understand and apply computers and software. After all, even though contemporary computing devices are among some of the most complex designs of mankind, the underlying essentials of logic, algorithms, programming, and computer construction are relatively simple as well as independent of computers themselves as physical devices. Consequently, United Kingdom, Estonia, and Finland have already adjusted their educational systems accordingly [LM14, pp. 62–66], and several other countries are following suit. In the future, programming skills will be considered as a part of people’s basic education, as for example reading, writing, numeracy, civics, and home science currently are [HAR14]. Moreover, computer science and programming studies are not strictly STEM (science, technology, engineering, mathematics) subjects any more: They can enrich learning of other subjects as well, and should be integrated¹ into and routinely used in other fields of study to give children an idea about the possibilities of their applicability in practice.

In Finland it has been possible to study elementary programming as an elective subject at some schools in grades 7–12, but from Fall 2016 onwards, it will be taught to everyone, starting from the first grade [LM14, pp. 48–49]. The first two grades will introduce precise algorithmic thinking through commands given between persons. During grades 3–6, concepts of programming will be studied using a visual programming environment (for example, *Scratch* [SCR]), and finally, from the seventh grade onwards, the visual programming will be replaced by some non-visual programming language, so that students will actually *write* programs instead of dragging graphical elements around the screen using for instance mouse, pen, or touch.

¹For example, the combination of both STEM subjects/disciplines, art, and design is dubbed as STEAM [e.g., EST14; STS].

To aid people—especially children—in the learning both programming and algorithmic/computational thinking, there are independent clubs and small-scale educational functions² for children to participate, too, and many parties are producing Internet-based course materials such as MOOCs (*Massive Open Online Course*) [SHE+14] and SPOCs (*Small Private Online Course*) [FOX13] as well as textbooks, board and computer games, robots, and the like³. Moreover, to help teachers responsible of primary and secondary education to acquire the skills necessary to teach programming, initiatives exist to provide training and self-study materials tailored specifically for them [e.g., KA; INN15; ERI+15].

However, there has been some controversy⁴ concerning for instance the actual necessity of coding and/or programming, the proper curriculum content, the methods to teach these skills for children with disabilities, ways to get more girls interested in computer science, and the need for updating teachers' skills related to information technology, especially algorithmic/computational thinking and programming.

There are also more practical concerns, like the computing devices and software to be used: Should educational institutions procure computing devices for all of their students, or should the students be allowed, encouraged or even required (even presuming that basic education is free) to bring their own devices and software along⁵, and whatever the answer is, what kind of devices and software to use. From the viewpoint of programming and computer science education, the obvious concern here is the availability of appropriate (educational/professional) development tools, which, in the case of embedded-systemish⁶ devices, is very limited or practically non-existent.

One common misconception among the objections against the programming education itself seems to be that every child is supposed to become a professional programmer, which in turn is easily deemed to be an unnecessary endeavour. To that end it should be emphasized that the idea of the programming studies as a part of the basic education is not to make a professional software developer out of everyone; that still requires university-level studies to begin with, not to mention the following life-long studying to keep up with new technologies, processes, and results of scientific research.

The true goal of the mandatory “programming” education is to introduce and accustom children to algorithmic/computational thinking and to give them a modern Everyman-level understanding of the functioning of computer hardware and software as well as their applicability in solving real-world problems. The children who have participated in this kind of education will be more capable of absorbing more advanced computing concepts encountered for instance in secondary and post-secondary education as a part of their later studies. *Coding*, or *writing program code* that looks like Greek—which may be the

²For instance: [AAL15B; OFF16; Kku; KKE; HEL15B; ELI15; SSS14; CCL; GIR15; KHA15B].

³Some examples. Internet-based materials: [TMI15; HEL15C; CoT15; COD15D; COD15E; COD15C; TEL14; NEU15; KHA15A; MD14; RYZ15]. Textbooks: [LIU; DRE14A; HEL15A; DRE15B; DIC15; DSM15; BRI13; MOR15; LEA13; MAR14A; DKP14A; DKP14B; BRE15; MCC14]. Board games: [PAA15; ROB14; COD14]. Computer games: [DRE14C; LIG15; HRM; HER16; COD15B; Foo; BBC15B; BBC15A; COD15A]. Robots: [WON15; DRE15A; PRI16].

⁴For instance: [AHO15; HAA15; LEH15; AAL15A; PÄI15; PAN14; GUZ14; CUR13B; CUR13A; KK15; MOU15; DRE15C; CUR15B; WAT15B; CUR15A; HIN15; HER15B; DRE15D; DRE14B; DOU15; DOU14].

⁵So called BYOD (*Bring Your Own Device*) trend [e.g., MCM15; JOH14].

⁶In the sense that the software selection available for them is limited due to for example their manufacturers' technical and licensing-based restrictions, user interface issues (for instance, a touch-based user interface versus keyboard and mouse), and performance issues (the more advanced the development environment is, the more computing power it will require to operate and offer its functionality). Examples include Chromebooks, which run Google's Chrome OS and are little more than an extension to Google's cloud-based services (which, among others, brings privacy concerns on the table), as well as tablets based on operating systems like Apple iOS, Google Android and Microsoft Windows RT.

only meaning some people associate with the term *programming*—is but an instrument to express and implement algorithms in practice, and even so, just a single approach for achieving that goal. However, *programming*, in the sense of *algorithmic/computational thinking*, is a much broader concept, and the “everything else” falling under it is the real essence of the education in question.

In addition to understanding the very essentials about computers and programming, the skill of writing simple programs with any programming language does not hurt, either. For example, in professional-level software suites it is common to provide programmable interfaces, which could be easily used to automate repetitive tasks and augment the software in question, if only the user understood enough to be able to read application programming interface (API) documentation and to write simple programs.

What is more, even if people would not program themselves, their jobs may require them to understand at least the basics about computer hardware and software. This knowledge is needed to be able for instance to utilize the technology they need to do their work; to help others and solve simple problems themselves without the help of IT support personnel; to discuss problems with IT support; to make decisions about purchasing software; and to discuss aspects of their own profession with software engineers implementing a new product to aid them in their work.

1.2 Introductory Programming Courses at Aalto University’s Department of Computer Science

One of the places educating the engineers of the future is Aalto University. It is a young institution that was established in 2008 and formed in practice by merging three Finnish universities in 2010 [A15B]. However, the oldest one of its three predecessors, the *Helsinki University of Technology* (HUT), has a much longer history that extends to 1849 [A13]—well beyond the invention of a microprocessor. While there was no Department of Computer Science at the time the first Finnish-built computer ESKO (*Elektroninen SarjaKOmpuutaattori*) was constructed at the then HUT⁷ during 1955–1960, computer science was being taught in several other departments since 1960’s [PAA13, pp. 35–38]. The first incarnation of a department of computer science was established in 1968, and after a few university-wide organizational metamorphoses during the years, the modern *Department of Computer Science* was formed in 2015 by merging three of its predecessors: The *Department of Computer Science and Engineering*, the *Department of Information and Computer Science*, and the *Department of Media Technology* [A15A].

The structure, the content, and the degree of difficulty of introductory computer science courses have obviously varied a lot during the years. For the students of computer science, there are currently four of these courses (Figure 1.1). The first two of those, *Programming 1* and *Programming Studio 1: Media Programming*, are designed to be taken together during the first Fall Semester, and the following two, *Programming 2* and *Programming Studio 2: Project*, during the following Spring Semester. The workload of these courses varies from 133 hours to 154 hours, and each of them is worth of 5 ECTS (*Eu-*

⁷As it happens, at the time ESKO was being constructed, HUT was located at Hietalahti, Helsinki [PAJ08, p. 430]. At the time of writing, those facilities are used by the Helsinki Metropolia University of Applied Sciences, and thus the writer studied his Bachelor’s Degree in them. In 2010, HUT, already moved to Otaniemi, Espoo, was merged into Aalto University. Esko itself was donated to the Museum of Technology in 1970 [MUS13; TEK15], at the premises of which it can be studied today.

Fall Semester	<table><tr><td>CSE-A1110 5 ECTS credits</td><td>Programming 1</td></tr><tr><td colspan="2">Workload 136 h<ul style="list-style-type: none">• Lectures: 6 h• Studying + doing exercises: 130 hContent<ul style="list-style-type: none">• Basics of OOP and FP• Stages of program code + corresponding tools• Program execution (e.g. call stack + garbage coll.)• GUIs, code quality, program design</td></tr></table>	CSE-A1110 5 ECTS credits	Programming 1	Workload 136 h <ul style="list-style-type: none">• Lectures: 6 h• Studying + doing exercises: 130 h Content <ul style="list-style-type: none">• Basics of OOP and FP• Stages of program code + corresponding tools• Program execution (e.g. call stack + garbage coll.)• GUIs, code quality, program design		<table><tr><td>ME-C2110 5 ECTS credits</td><td>Programming Studio 1: Media Programming</td></tr><tr><td colspan="2">Workload 133 h<ul style="list-style-type: none">• Lectures: 2 h• Group teaching + preparation: 20 h• Exercises + doing exam: 110 h• Course feedback: 1 hContent<ul style="list-style-type: none">• Basic concepts of digital media• Programmatical processing of images and sound</td></tr></table>	ME-C2110 5 ECTS credits	Programming Studio 1: Media Programming	Workload 133 h <ul style="list-style-type: none">• Lectures: 2 h• Group teaching + preparation: 20 h• Exercises + doing exam: 110 h• Course feedback: 1 h Content <ul style="list-style-type: none">• Basic concepts of digital media• Programmatical processing of images and sound	
	CSE-A1110 5 ECTS credits	Programming 1								
Workload 136 h <ul style="list-style-type: none">• Lectures: 6 h• Studying + doing exercises: 130 h Content <ul style="list-style-type: none">• Basics of OOP and FP• Stages of program code + corresponding tools• Program execution (e.g. call stack + garbage coll.)• GUIs, code quality, program design										
ME-C2110 5 ECTS credits	Programming Studio 1: Media Programming									
Workload 133 h <ul style="list-style-type: none">• Lectures: 2 h• Group teaching + preparation: 20 h• Exercises + doing exam: 110 h• Course feedback: 1 h Content <ul style="list-style-type: none">• Basic concepts of digital media• Programmatical processing of images and sound										
Spring Semester	<table><tr><td>ICS-A1120 5 ECTS credits</td><td>Programming 2</td></tr><tr><td colspan="2">Workload 135 h<ul style="list-style-type: none">• Lectures: 24 h• Group teaching: 36 h• Independent work: 72 h• Exam: 3 hContent<ul style="list-style-type: none">• Computer architecture and program execution from gate level to high-level progr. languages• Computational resources + measurement• Interfaces + recursion• Basics of algorithm design and analysis</td></tr></table>	ICS-A1120 5 ECTS credits	Programming 2	Workload 135 h <ul style="list-style-type: none">• Lectures: 24 h• Group teaching: 36 h• Independent work: 72 h• Exam: 3 h Content <ul style="list-style-type: none">• Computer architecture and program execution from gate level to high-level progr. languages• Computational resources + measurement• Interfaces + recursion• Basics of algorithm design and analysis		<table><tr><td>CSE-C2120 5 ECTS credits</td><td>Programming Studio 2: Project</td></tr><tr><td colspan="2">Workload 154 h<ul style="list-style-type: none">• Lectures: 32 h• Exercises + self-study materials: 42 h• Project: 80 hContent<ul style="list-style-type: none">• OOP-based program design• Object-oriented abstractions• Integrated development environments• Unit testing• Basics of concurrent programming</td></tr></table>	CSE-C2120 5 ECTS credits	Programming Studio 2: Project	Workload 154 h <ul style="list-style-type: none">• Lectures: 32 h• Exercises + self-study materials: 42 h• Project: 80 h Content <ul style="list-style-type: none">• OOP-based program design• Object-oriented abstractions• Integrated development environments• Unit testing• Basics of concurrent programming	
ICS-A1120 5 ECTS credits	Programming 2									
Workload 135 h <ul style="list-style-type: none">• Lectures: 24 h• Group teaching: 36 h• Independent work: 72 h• Exam: 3 h Content <ul style="list-style-type: none">• Computer architecture and program execution from gate level to high-level progr. languages• Computational resources + measurement• Interfaces + recursion• Basics of algorithm design and analysis										
CSE-C2120 5 ECTS credits	Programming Studio 2: Project									
Workload 154 h <ul style="list-style-type: none">• Lectures: 32 h• Exercises + self-study materials: 42 h• Project: 80 h Content <ul style="list-style-type: none">• OOP-based program design• Object-oriented abstractions• Integrated development environments• Unit testing• Basics of concurrent programming										

Figure 1.1. Essential information concerning current Scala-based introductory programming courses of Aalto University's Department of Computer Science aimed at students majoring in computer science.

ropean Credit Transfer and Accumulation System) credits. All in all, there are software development studies worth of 20 ECTS credits during the freshman year of CS students⁸.

As can be seen from the divisions of workload in Figure 1.1, these four courses are designed in different ways. The first two of them, *Programming 1* and *Programming Studio 1: Media Programming*, have similarities to the *flipped classroom* concept [e.g., COL13; MAR14B; VUK14; BS15; WAT15C], in which theory is studied independently outside of the classroom while the classroom time available is used for practical applications, guidance, and exercises. There are two versions of the *Programming 1* course: An in-house version, the workload and teaching methods of which are described in Figure 1.1, and a MOOC version, which does not include the lectures and exercise groups but is available for everyone interested in learning the basics of programming. As for the *Programming Studio 1: Media Programming*, there is no MOOC but the course still relies on self-directed studying and using the classroom time for group-based teaching and doing exercises. The latter two courses, *Programming 2* and *Programming Studio 2: Project*, are more traditional in that while they will partly utilize Internet-based materials, they still have regular lectures. The material of the *Programming Studio 2: Project* is also available as a MOOC that carries on from where the *Programming 1* course ended.

While some of the previous programming languages used in these courses include for instance Java and Python, the chosen language for the four courses described above is Scala. It is a versatile professional statically-typed programming language supporting both imperative object-oriented programming (OOP) and functional programming (FP) paradigms. Scala is well-suited for an introductory programming language for many

⁸For completeness: There are also two Python-based introductory programming courses aimed to the students of other disciplines than CS; these courses will not be discussed in this thesis.

reasons. For example, it is possible to start from very simple concepts and gradually teach more and more advanced concepts of both OOP and FP while still using a single language. The syntax is closely related to Java, so the threshold to learn it is relatively low especially for those that have formerly programmed with Java. Furthermore, many concepts of other programming languages such as Python, C++, and JavaScript, can also be applied to Scala. From the opposite viewpoint, programming concepts learned using Scala are applicable to many other programming languages, so that the benefits of any work done will not be limited to these four introductory courses. The features making Scala a novice-friendly language include for instance a pure object-based type system; garbage-collection instead of manual memory management; an extensive collection type framework included into the standard library; and a REPL (*read-evaluate-print loop*) that enables fast experiments and demonstrations to introduce the language and any class libraries.

The primary study materials of these four courses are implemented as web sites (of which two are available as MOOCs, as described above), and occasionally some lecture slides can be used, too. Some of these materials evolve rapidly, as they are being constantly improved even during the courses. The software used during the courses obviously includes a recent Scala release, and because the Scala implementation used targets and runs on the Java Virtual Machine (JVM), a compatible Java Development Kit (JDK) is also required. The integrated development environment (IDE) supported by the course personnel is the Eclipse-based Scala IDE [SID15], but other IDEs, like NetBeans [NTB] and IntelliJ IDEA [JET15], are also possible (see § 4.3). The REPL functionality, available through the operating-system-provided command prompt and terminal windows as well as through IDE integrations, is also utilized. Some of the exercises require students to download exercise templates and additional libraries. Automatic assessing is utilized in the extent possible, but some of the exercises require manual evaluation. In addition, project work is naturally assessed manually by instructors and assistants.

1.3 Research Questions, Goals, and Process

As told in the beginning of this chapter, this thesis is concerned with Media Programming (§ 3.2)—that is, the idea of contextualized teaching (§ 2.2) while using programmatical creation and processing of multimedia as the context. Probably the most well-known example of this approach is its superordinate called *Media Computation* (Chapter 3), which has been developed at Georgia Institute of Technology [GUZ13] from 2002 onwards. As described in the previous section, among the freshmen at Aalto University’s Department of Computer Science, Scala is the first programming language encountered by people, who have not programmed before. It is being used at least through the freshman year to teach both basics and more advanced CS concepts. To make these courses more interesting and illustrative for students, the personnel of the *Programming 1* course described above would like to take advantage of Media Programming as a context on their course.

Unfortunately, as far as is known, decent Scala-based tools to enable applications of multimedia-oriented context have been conspicuous by their absence. Several code libraries and stand-alone tools to apply this approach (§ 3.5) have emerged during the past 12 years, but they are suitable and practical only with other programming languages, such as Java and Python. This thesis intends to improve this situation by (1) carrying out research to find out answers to four research questions and (2) achieving one research goal; these are presented in Table 1.1 below.

Table 1.1. Research questions and goals.

ID	Description
RQ1	What kind of tool-based contextualization approaches exist for different programming languages?
RQ2	Considering the tools of RQ1 that support Media Programming, how do their user interfaces support displaying multimedia?
RQ3	What genres and exercise types exist in Media Programming?
RQ4	Considering utilization of Media Programming at Aalto University's Department of Computer Science in Scala-based learning environments, what potential technical challenges exist?
RG1	Development of a prototype that fulfills the requirements defined in the § 4.2.

Both **RQ1** and **RQ2** are important not only to better comprehend the current practice and tool selection, but also to contribute to the decision on the best way to design the prototype of **RG1**. The answer to the **RQ3** helps not only (1) to comprehend the current practice, but (2) to give a solid basis with examples and exercises for building a Media Programming curriculum. As for **RQ4**, the results help to make decisions on whether or not to spend further effort on development related to Media Programming, and if so, how to channel them the most profitable way.

The prototype of **RG1** has three purposes: (1) Observations during its development process provide answers to the research question 4, (2) it allows exploration of possibilities to apply multimedia-oriented context to Scala-based introductory programming courses, and (3) the prototype itself may be improved to serve as a tool for improving Scala-based introductory programming teaching in practice. Thus, there are mutual dependencies among the research questions and goals: Except the obvious ones between the research questions 1 and 2 as well as between the research question 4 and the research goal 1, also the third research question is indirectly dependent on both the first and the second question.

What comes to the results' generalizability, only the results of **RQ1**, **RQ2**, and **RQ3** will be context-independent knowledge (in the sense of the context in which the research is carried out). The results of **RQ4** will obviously be (1) related to Scala-based teaching, and (2) primarily context-dependent, because the requirements and implementation of the prototype are affected by for instance the current needs of the courses of § 1.2 as well as personal opinions and skills of the course staff and the writer of this thesis (the implementor of the prototype). On the other hand, these results may still be partly generalizable to any context having the same characteristics, tools, etc. than the prototype and the courses just mentioned.

The research process for this thesis has three phases, as described below:

- I: The first phase consists of finding answers to **RQ1**, **RQ2**, and **RQ3**. They will be formulated on the basis of both a descriptive literature review targeted to any material available about Media Computation and a practical trial of the software tool selection currently available (see Chapter 3). The search engines and databases used to collect literature are listed in Appendix A.

- II: The second phase focuses on both [RQ4](#) and [RG1](#), and comprises building a research prototype (see Chapters 4 and 5) that reflects the first phase’s tools’ characteristics that are found suitable to be used in Scala-based introductory programming courses. These characteristics are agreed on in conjunction with the instructors of the *Programming 1* course.
- III: The third phase equals carrying out a pilot study concerning applicability of the prototype produced in the second phase (see § 6.2). Due to the limited time frame of this work, a full-scale research on students is out of the question (see the next section). Instead, the applicability will be assessed from teacher’s point of view—that is, whether or not the teachers are able to design demonstrations and exercises they find pedagogically useful (see also § 4.2 and [R9](#)). Consequently, the study will be carried out on some of the teachers responsible for any relevant courses, such as the ones described in the previous section.

The following section describes some general restrictions that apply to this process.

1.4 Scope Restrictions

The scope of this work is somewhat restricted. The main reasons behind the constraints are the following:

- The educational role of this work is master’s thesis, which both constrains for example the nature of the work done and sets quite a narrow time frame for its completion.
- As the writer of this thesis has been forced to accept during the work done for it, CER is such a multifaceted discipline that gaining a complete understanding of even a small slice of it seems to be impossible in the course of master’s thesis work.
- The prototype is of such a nature that both its development and the related research could continue indefinitely.

Originating from the foregoing reasons, the following general restrictions will be put on the record here:

- The main purpose of this thesis is not to provide results for instance to any educational or psychological aspects related to it. Some literature concerning education and psychology is used in the following two chapters to provide basic background information for the work done, but otherwise the educational and psychological viewpoints and literature are basically out of scope.
- In the context of contextualizing introductory programming courses, there exist articles concerning both Media Computation and contextualized teaching in general. However, only a few of them will be included in the scope of this thesis.
- This thesis concerns itself only with the very core of the current practice of Media Computation; this area will be defined (§ 3.2) to be called as Media Programming.

- **RQ1** and **RQ2** : The software tested for this thesis has to be executable and usable in contemporary general-purpose computers running Microsoft Windows 10 Pro or the latest freely available GNU/Linux distributions, and they (or at least their demo versions) must be freely available—usually as downloads through the Internet.
- **RQ2** : The scrutiny of ways to utilize multimedia focuses on usage of bitmaps; closer examination related to other multimedia types is out of the scope here.
- **RQ3** : The scrutiny of genres and exercise types of Media Programming is restricted to the Media Computation curriculum of Guzdial & Ericson.
- **RG1** : The prototype is developed to be extensible, but its current functionality is restricted to rudimentary bitmap processing. While Media Programming, Media Computation, and multimedia are much broader concepts, their full coverage in the depth desired was out of question in the time frame of this work.
- **RG1** : Because of the limited time frame of this work, the prototype cannot be subjected to long-term scientific research that concerns for example the learning results or drop-out rates of the relevant courses at Aalto University and includes proper test and control groups selected from the population of students participating to those courses. Instead, as already described, a tiny evaluation is done from the teachers' perspective in the form of a pilot test among the teachers of the relevant courses.

Other restrictions are indicated along the way as they are run into.

1.5 Existing Literature

This thesis introduces Media Programming through presentation of some background information related to both motivation in educational setting and contextualized teaching. Some of the motivation-related articles and textbooks [e.g., LP16; MAY14B; AD11; BKK06] cite scientific research more than others [e.g., PRI14; BT11], and one of them [PIN04] popularizes motivation-related scientific research clearly from the writer's own viewpoint, also including some personal opinions. Contextualized teaching in general is discussed in a few articles [PH14; MFR13; CC10; Guz10] and in the motivation-related literature mentioned above. Examples of applied contextualization in the context of computer science and introductory programming courses are given in many articles⁹, including those dealing with Media Computation. Some context for Media Computation is drawn also from two textbooks [FJ14; MAY14A].

About Media Computation, and thus about Media Programming, there exists a bunch of articles¹⁰ and a book chapter [Guz15, pp. 53–68] at least mentioning the approach or describing it, its history, as well as some results achieved with it. A few textbooks¹¹ offer a view to the actual course content. Many of these resources are at least co-authored by Mark Guzdial, who also discusses Media Computation in his blog, but some third-party accounts, such as the one of Simon et al. [SIM+10], exist as well. Furthermore, Media

⁹For instance: [KAF+14; KAF+13; COW+12; KB12; RIL12; TEW+08].

¹⁰For instance: [DEN14; Guz13; LEE13; POR+13; PS13A; PS13B; Guz+10; LH10; SIM+10; ST08; YG08; GF05; TFG05; RPG04; Guz03; GS02].

¹¹For instance: [GE13; GE11; DCE10; GE05].

Computation has brought about some variations, such as Mediascripting [RDW13], generative art [GKX12], real-time video [CP05], Media Propelled Computational Thinking [FRE+10], and Computational Remixing [FMV15; FRE+14; MAG+13].

The number of Scala-related papers is low, and publications that discuss Scala as an introductory programming language seem to be practically non-existent (excluding for instance workshops [e.g., LLT14], panel discussions [e.g., BBN10] as well as publications merely mentioning Scala as an example [e.g., HER15A; IVA+15] or listing statistics such as adopters of a given language [e.g., FAC15; SSA15]). Even the only somewhat relevant publication [BLA11] is a short conference paper reporting experiences—not discussing scientific research carried out on the subject.

Finally, about the central topic of this thesis—the combination of Scala and Multimedia Programming—there seems to be only one article [BH] written, and it seems not to be formally published anywhere.

1.6 Computing Education Research

Computing education research (CER) is a relatively new discipline—it has been around only for several decades. One of the early efforts related to it is from 1960s, at which time researchers at Massachusetts Institute of Technology (MIT) and Raytheon BBN Technologies were already exploring possibilities to teach children to program using a specially-designed programming language called *Logo*¹² [GUZ04, p. 129; SOL]. They also had physical robots capable of moving around and drawing images on paper with an attached pen according to a given Logo program. The robots were doubled as a concept of a turtle—an image on a graphic screen—which was programmable both to move around and to draw images on the basis of its movements, as its physical counterparts did. Judging by today’s standards, the computers of the time, such as Digital Equipment Corporation’s (DEC) PDP-10, were huge, expensive, rare, and *very* slow, and were incomprehensible for everyone but those having an appropriate technical training. However, that did not stop scientists from studying their educational applicability—not even where children in preschool were concerned.

As a discipline, CER is broad and nuanced. It is centered around education in the context of computing, and employs theories, methods, and results of other useful disciplines as instruments to produce, support, justify, explain, and develop its results. Thus it could be characterized as a hybrid of computing, education, and psychology (e.g., educational psychology), mathematics (e.g., statistics), as well as for instance anthropology (e.g., ethnography). Its purpose is to develop and collect both *content knowledge* and *curricular knowledge*, of which the former refers to knowledge of a specific content from teaching perspective, whereas the latter denotes the approaches available for teaching a specific subject [SOR12, p. 11]. Moreover, it is concerned with both the ways people learn computational concepts and how to aid and improve those learning processes [INT15].

Education is, as described above, in the very core of CER. As a discipline and research area, it touches many different disciplines in itself, such as philosophy, history, sociology, and psychology [BB12, pp. 8–9]. The philosophical angle includes aspects such as nature of education, how to organize knowledge, and what it means to be an educated person. The historical viewpoint is concerned about relationships and causality between social, political, and educational development, and contributes to understanding the past and

¹²Logo itself—as ported and modernized versions, of course—is still being used [e.g., STO+13; BEL; LBR; SRF; TAC]. It also directly affected to programming languages such as Boxer and Smalltalk-72 (the first object-oriented programming language) [GE11, pp. 97–98], and through the latter, to several other (in 2016, more modern) programming environments [e.g., KOG; SCR; Sq].

the present as well as to planning of future strategies. The sociological perspective explores for instance social ideologies and structures as well as effects of gender, ethnicity, social class, and economical status. The most interesting standpoint concerning CER, however, is the psychological one. It concerns itself for example with learning processes, intelligence, personality, individual learners as parts of groups, as well as motivation and achievement.

In practice, CER has a number of research areas, two accounts of which will be presented in the next section. Traditionally, publications of these areas have biased to the kind that lacks scientifically valid research and merely reports experiences and stories. These so called *practice papers* [e.g., FP04B, pp. 2, 79–80; PM09; SIM15, pp. 17–18], or “Marco Polo papers”, are useful on practical level, but their scientific value is questionable especially in the eyes of other research disciplines that have stricter and more established conventions. During more than the past ten years, there has been discussion for example on advantages and disadvantages of that kind of papers, what scientific research in the discipline of CER means in the first place, and what research methods are suitable in the CER context. In the long term, at least conferences and scientific journals have the power to try to steer these conventions by adjusting their requirements for papers they accept and by introducing new categories and tracks for different kinds of papers.

The researchers in CER share one very powerful characteristic, which touches also this thesis: Probably most of the people working in the field have at least some software development training, and anyone having a Master’s Degree in CS has a basic training of a full-fledged software engineering professional. This means that the discipline has a great potential to produce both research prototypes and working educational software, which otherwise might never get developed. Additionally, if the software developed is usable, robust, and generalized enough, it may spread to other educational institutions and disciplines, and even spawn commercial products and services [e.g., AG16].

1.7 Classifications and Relations to Research Disciplines

To begin with, one of the most essential literature resources in the field of computing is probably Association for Computing Machinery, Inc. (ACM), which maintains a classification for its publications. The last revision [ASS12] is from 2012 and contains approximately 2000 classes to assign to documents. Still, finding a proper class for this thesis seems to be problematic. The classification has several categories for software tools, but none concerning educational ones. There are also two categories for educational content, *Applied computing* → *Education*, as well as *Social and professional topics* → *Professional topics* → *Computing education*, but as already said, none of the subclasses of these two categories describe the essence of this work—the tool developed for an educational purpose. The latter one has several subclasses touching some possible utilization areas of this work, but none of them describes its very essence. None of these upper-level classes cannot be assigned, for that would require the work to cover at least the most of their subclasses. Probably the nearest match would be *Social and professional topics* → *Professional topics* → *Computing education* → *Computing education programs* → *Computer science education* → *CS1*, in the meaning of introductory programming courses, although this work is not directly concerned with those courses themselves but a tool that could later be utilized on them.

In terms of scientific research disciplines, the rough location of this work would probably be inside the field of computing education research (CER) discussed in the section before. In the light of the trend in CER to emphasize research instead of reporting anecdotes, the inability to carry out a proper evaluation of the prototype’s effects is a shame: There are no proper hypotheses, statistically valid tests, nor rejections of false theories. Consequently, this thesis is not a very scientific one, and the essence of it could classify as a typical example of a practice paper (in distinction from *research papers*), and more precisely, a *tools paper* [e.g., MAL14].

One way to visualize the research disciplines intertwining under CER is given by Sorva [SOR12], who places his work in a Venn diagram containing three sets: computing, education and psychology. However, that approach is not so useful in this case due to two reasons: First, as the main contribution of this thesis is a software tool, the thesis could be placed under the discipline of computing alone. On the other hand, the concepts and theories that justify the possible usefulness of the prototype, draw heavily from education and psychology—this makes the exact location of this work unambiguous. Second, the point of interest here are not the exact overlappings of the research disciplines but rather the relationships between them and this thesis. For these reasons, Figure 1.2 below visualizes the relationships of this work in a different way.

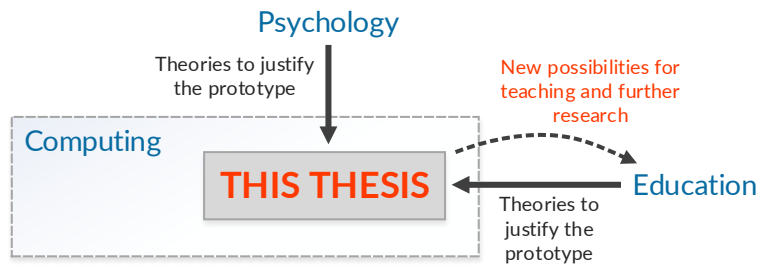


Figure 1.2. Relations between this thesis and the disciplines of computing, education, and psychology. As can be seen, this thesis belongs in the discipline of computing, it draws theories from psychology and education, and possibly offers educators and researchers new ways to teach as well as new possibilities for research.

Considering the disciplines of research at the level of Figure 1.2 is one thing. To get a more detailed picture of the relationships between this thesis and its surroundings, it is fitting to consider the research areas under CER. During the existence of CER, researchers have spent efforts to identify and define them. One of those contributions is that of Fincher & Petre, which identifies ten different areas: *Student understanding*; *Animation, visualization, simulation*; *Teaching methods*; *Assessment*; *Educational technology*; *Transferring professional practice into the classroom*; *Incorporating new developments and new technologies*; *Transferring from campus-based teaching to distance education*; *Recruitment and retention*; and *Construction of the discipline* [FP04B, pp. 3–7].

Nevertheless, the world changes, research advances, and new research areas emerge while older ones may become saturated. Consequently, the classifications and taxonomies used to analyze, label, and understand research subjects, must also remain up to date. At the time of writing, probably the most recent proposal for a classification to classify computing education papers is published by Simon [SIM15, pp. 93–94]. Although the classification does not directly concern research areas themselves, the 21 themes it presents resemble those of Fincher & Petre and—expecting the theme list to be extensive—probably

illustrate the currently known areas of CER quite well. The list contains all the areas of Fincher & Petre, some of them broken into different themes while introducing several new ones. Figure 1.3 takes advantage of those themes as research areas by placing this thesis among them to scrutinize its relationships.

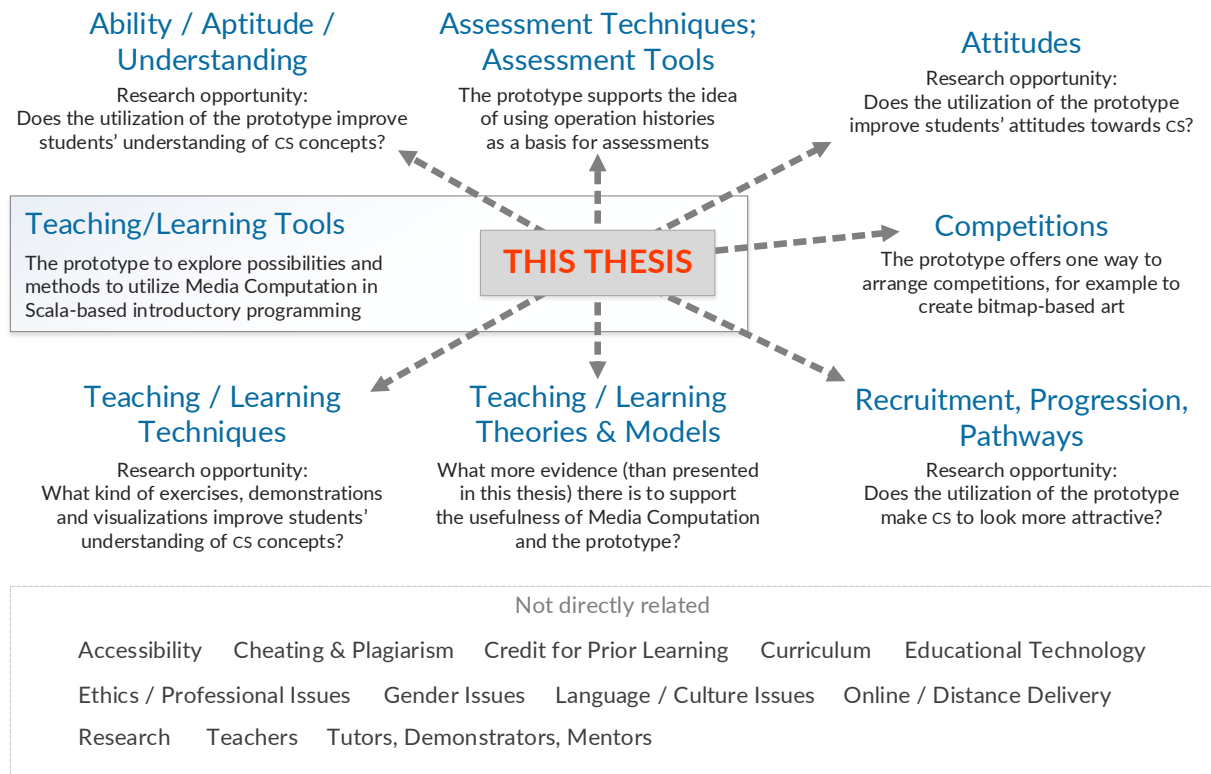


Figure 1.3. Relations between this thesis and CER areas. The areas considered here are those identified by Simon [SIM15, pp. 93–94] as themes of CER papers.

As Figure 1.3 illustrates, the essence of this thesis naturally belongs in the area of *Teaching/Learning Tools*. In addition, there are possible relations to eight other CER areas, and the unrelated 12 areas are listed as well. The research opportunities mentioned in the figure are examples; for some of the areas, there are also other possibilities available.

1.8 Content and Notations

After this introductory chapter, the presentation in this thesis continues as follows: Chapter 2 discusses matters of this work's theoretical background, namely, motivation and contextualized teaching. It is followed by a discussion of the Media Programming approach and short introductions to some software tools that are or could be used with the approach (Chapter 3). After these, the following two chapters concentrate on the research prototype developed for this thesis: Chapter 4 presents its functional side, while Chapter 5 focuses on its implementation and technical aspects. Chapter 6 deals with evaluation of the prototype as well as this thesis as a whole, and finally, this thesis is concluded in Chapter 7.

Furthermore, there are some appendices to supplement this work. Appendix A contains a list of the data providers used to search literature for this work, and some program

code examples which were used to produce some figures presented in § 3.5 are listed in Appendix B. Finally, Appendix C presents theme-based listings of some resources related to the topics of this thesis.

This thesis contains lots of figures, some of which are diagrams. A portion of these diagrams is not based on any methodology, whereas some comply with the Unified Modeling Language (UML) 2.4 standard. The content and applications of the standard are expected to be known, and will not be discussed further in this thesis. Some of the other figures are image manipulation examples, the source bitmaps of which are presented in Figure 1.4 below.

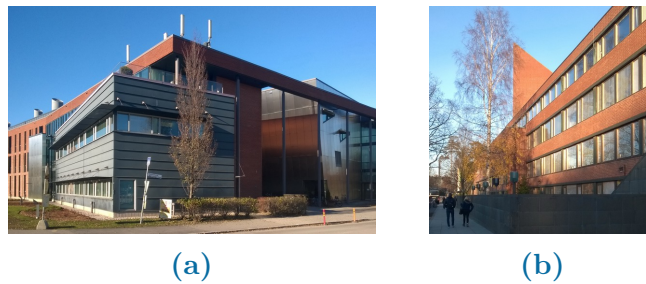


Figure 1.4. The bitmaps that will be used as source images throughout the thesis.

The thesis also utilizes several ways to typographically accentuate specific kinds of content; these notations are listed in Table 1.2 below.

Table 1.2. Typographical notations used in this thesis.

Notation Example	Description
§ 99.9	Section in this thesis.
R99	A requirement definition.
RG99 / RQ99	Research goal / question.
Student	Names of stakeholders of the prototype.
BitmapOperationList	Program code, an identifier related to it, or command line content.

It is also useful to notice that in the electronic version of this document, many references are actually hyperlinks. These references include chapters, sections, figures, tables, research questions, citations, and requirement definitions. The table of contents and the lists of figures, tables, and code examples that follow it, are hyperlinked to the referenced content.

Chapter 2

Motivation and Contextualization in Education

The first chapter introduced the reader to this thesis through its background, research questions and methods, content, and so on. This second chapter concerns itself with theoretical background of this thesis, such as theories of motivation (§ 2.1) and contextualized teaching (§ 2.2). As a conclusion, this chapter answers fully to **RQ1** and essentially to **RQ2** by presenting a classification of contextualization approaches and software tools used with them (§ 2.3), with a special focus on the ways the user interfaces of tools supporting the Media Programming approach (§ 3.5) use multimedia. The next chapter will continue from there by concentrating on the Media Programming approach itself.

2.1 Motivation

Biggs & Tang state that “there is no such thing as an unmotivated student; all students not in a coma want to do *something*” [BT11, p. 34]. They also describe three levels of professional proficiency for teachers [BT11, pp. 16–20]. At the first level, the teacher’s way of teaching is constant and based on transmitting information to the students. Teacher at this level focuses on *what the student is*, thinking that some students are good or motivated while on the other hand some others are unmotivated or “just do not get it”; Biggs & Tang call this as a *blame-the-student* approach. The second level of proficiency is teacher-centric: Teachers concentrate on *their own actions* and teaching methods, but their teaching is still based on transmission, and the characteristics of students as learners are still ignored.

Only on the third level of Biggs & Tang, the teaching becomes student-centric—it focuses on *what students should do* to learn. Teachers know for example what the intended learning outcomes are and what students should do to achieve the level of understanding specified in the learning outcomes. Teachers become facilitators who help the students to learn themselves by whatever means appropriate for the content and the student population in question. It is also part of the teachers’ job to try to motivate their students towards active learning.

To provide theoretical background for Media Programming, this section presents several theories related to the motivation of students. As motivation in its own right is a complex topic with several application areas (such as study versus work motivation), only a small peek into it can be presented here.

Maslow's *hierarchy of needs* [MAS43] will service as a point of reference for the other theories discussed in this chapter. It represents the basic needs of human beings by using five levels that are often represented as a pyramid. These levels in the bottom-up order are *physiological needs*; *safety*; *love, affection, and belongingness*; *esteem*; and *self-actualization*. The basic idea of those levels is that for a specific need to take control, the lower-level needs must be satisfied first. However, Maslow points out that (1) the order of the needs is a generalization which is not the same for all people, and (2) the needs of a single level do not have to be fully satisfied before the upper-level needs emerge. Consequently, all of the levels may be in effect at the same time with different degrees of satisfaction for each individual observed. Maslow also emphasizes that a certain behavior can be motivated by several of the needs at the same time, and that all behaviors are not determined by the basic needs at all.

Behaviorism [e.g., PR14, pp. 7–10; AD11, pp. 220–221]—the earlier and the other of the two major viewpoints to learning (in addition to constructivism)—does not deal with motivation as a conscious asset. It explores learning through individual's extrinsic actions, and only through them—no intrinsic information processing that the individual does is acknowledged to exist at all. According to behaviorists, there are two ways for someone to become “motivated”: classical conditioning and operant conditioning. The former case studies behavior of subjects after a certain stimulus, whereas the latter is concerned with effects of rewards (or punishments) given after a certain behavior.

Classical conditioning has happened when a stimulus causes the learner to react in a certain way (in principle) every time it is repeated. The stimulus is *unconditioned* when it is the meaningful phenomenon itself. An example of this could be an animal, which salivates when it sees food. On the other hand, a reaction to a stimulus that the learner has associated with the meaningful phenomenon denotes a *conditioned* stimulus. The classic example of the latter are Ivan Pavlov's dog experiments, where the sound of a bell induces salivation on dogs that are conditioned to relate the sound to food given to them. Generally, for instance the drilling and other practical battle/emergency training of airline pilots and flight attendants, military, police, security, and martial artist organizations aims to condition the trainees to certain stimuli, so that in life-endangering situations they can react to them with a split-second latency and thus possibly prevent damage for both themselves and any third-parties involved.

Operant conditioning is a carrot-and-stick approach to motivate the learner [PIN04]: It views learning in the light of rewards and punishments that are purposely used to positively or negatively reinforce a certain behavior. Critique about using external rewarding and punishments exist [e.g., PR14, pp. 11–15, 17; AD11, pp. 227–228; BT11, p. 35; PIN04], and as Anderman & Dawson point out [AD11, p. 221], individuals' cognitive processes are able to override any previously-learned behavior. Rewards and punishments may work in the case of dull and strongly repetitive work¹ or in shaping acceptable behavior in school. For example, from the viewpoint of students, the purpose of all assessments; extra rewards like stamps, stickers, and public praise; and punishments like admonitions, detentions, and even temporary suspensions, are intended to shape students behavior by reinforcing the good customs and by extirpating the harmful ones.

Attribution theory [e.g., LP16, p. 96], as the other theories presented in this chapter, is part of *cognitive psychology*, which is concerned with mental processes such as learning. Its basic idea is very simple: People have more motivation to understand phe-

¹A stereotypical example is a Taylorian assembly line: Frederick Winslow Taylor's *Scientific Management* from the end of 1800s treats workers basically as robots that act as tiny independent parts of a complicated manufacturing processes and are incapable to understand the greater purposes behind even the most simple tasks [HUM91].

nomena (especially negative or unexpected ones) if they have first-hand experience of them. The motivation of a subject is estimated on the basis of (1) the degree in which the individual feels control over the phenomenon, (2) the phenomenon's degree of persistence, and (3) locus (external versus internal) of the phenomenon. For example, if a subject fails a specific endeavor, such as a test or a sport performance, they want to understand why the failure took place. Was it their own fault, or was there some external factor preventing them from succeeding in the challenge? How likely it is that the same problem occurs again in the future? How much control the subject feels over the offending phenomenon? Is it possible to prevent the problem at least in some degree in the future by doing something reasonable, or is it beyond the subject's abilities and influence? The answers to these questions affect the subject's future attitudes and feelings towards similar challenges, and by influencing the causes of the failure, it may be possible to improve the subject's future attitudes and for instance to prevent the subject from dropping out from a course or a school.

Self-determination theory [e.g., LP16, pp. 93–94; AD11, pp. 225–226] classifies the degree of individual's motivation on the following six-step scale: *amotivation*, pure/introjected/identified/integrated extrinsic motivation, and intrinsic motivation. *Pure extrinsic motivation* is based solely on a reward expected to be received, whereas *Introjected extrinsic motivation* is caused by a subject's feelings, for instance about social acceptability. *Identified extrinsic motivation* means that a subject values a certain task to some extent for example due to expected skill level increment, even if the task itself is not pleasant. Finally, the most internalized class of extrinsic motivation, *Integrated*, refers to a state where a person has accepted a task to be part of themselves while they have not a specific interest in the task itself. The motivation is described to deepen by increasing person's needs for relatedness (Maslow's third level), autonomy, and competence (cf., Maslow's fifth level).

The concepts of intrinsic and (pure) extrinsic motivation [e.g., LP16, p. 94] are vital as a coarse-grained classification of motivation source. Intrinsic motivation is found to be linked for example with creativity as well as academic engagement and achievement. In contrast, an external duress as a source of motivation may appear for instance as inappropriate strategies and attitudes towards learning, anxiety, relatively weaker capacity to overcome challenges, suboptimal academic achievement, and even school drop-outs.

Expectancy-value theory [e.g., LP16, pp. 92–93; AD11, p. 226] tries to approximate reasons for *why* people would be motivated and why they would choose one task over another, in contrast to self-determination theory that classified prevailing levels of motivation itself. Expectancy value consists of four parts: *Utility value*, which refers to task's usefulness for other aspects of person's life; *attainment value*, which means the subject's personal desire to succeed; *intrinsic value*, which approximates the amount of enjoyment an individual receives by doing the task in question, and finally, *cost*, which encompasses all negative aspects that performing the task might cause to the subject. These four indicators could be determined for each task to be considered both to understand them better as themselves and to compare them to each other.

Self-efficacy [e.g., LP16, pp. 91–92; AD11, pp. 224–225] is a concept from *social cognitive theory*, and as implicated above, it refers to individual's beliefs about their own possibilities to finish an undertaking, such as to learn or master a skill. These beliefs in turn affect to the choices made by students. They can be increased by direct and vicarious experience of mastery as well as social persuasion and encouragement, whereas they can be diminished by various physiological reactions, such as tremors, petrification, and clumsiness out of feelings of nervousness, fear, and shame. Furthermore, the per-

ceived mastery can be furthered by helping learners to recognize their progress as well as by providing constructive feedback and suitable *cognitive scaffolding* [BKK06, p. 477]. Generally people are motivated by individually appropriate difficulty level (not too hard but not too easy, either) [BKK06, p. 481; PIN04]; it can be reached by either choosing a more difficult task or reducing the challenge with a proper scaffolding. A suitable difficulty level may also increase both intrinsic value and expectancy value associated with a task by the learner. In addition, collaboration in a group may also increase the feeling of mastery if the combined skills of the group are broader than those of any member alone.

Self-efficacy theory is related to two intelligence-related concepts that Pink calls *entity theory* and *incremental theory* [PIN04]. The former refers to the individual's belief about intelligence being an entity—a fixed quantity of an individual—and that nothing can be done to improve it. On the other hand, the latter describes a belief that one's intelligence can be developed, or increased, by exercising it. Thus, those subscribing entity theory see challenges as measurements and possibly personal humiliations, whereas the opposite camp think them as opportunities for personal growth. Similarly, these standpoints affect to persons' choices about the tasks they engage themselves.

Achievement goal theory [e.g., LP16, pp. 94–95; AD11, pp. 223–224, 228–230; PIN04] encapsulates an idea of persons' attitude towards a task (1) when they feel high self-efficacy, have the incremental view of intelligence, or are already inwardly motivated, versus (2) when their self-efficacy is low, they subscribe entity theory, or they lack the intrinsic motivation. The former case portrays the completion of a task a person is facing into a *mastery goal* (such as “learn Greek because it is a fascinating language”). In this case, a person is interested in mastering the task for the sake of the mastery itself, and compares their accomplishments to only themselves—not to others. Then again, in the latter case the task will more likely be seen as a *performance goal* (such as “get an A from that Greek test tomorrow”), which means that an individual is not (primarily) interested in the actual task but instead tries to appear competent compared for example to their peers or idols, often by spending minimal effort on the task itself. These two modes of action are also known as *intentional learner* and *schoolwork module*, respectively [BKK06, p. 478]. In addition, Biggs & Tang describe four factors for feeling a task to be worth your while [BT11, p. 35], and only one of them, the *intrinsic motivation*, leads to the mastery goal approach. The three other factors—*extrinsic motivation*, *achievement motivation*, and *social motivation*—lead to the performance-oriented approach.

The model of the two action modes presented above can be further augmented into a 2-by-2 matrix by attaching word “approach” to their names and adding two other modes that are labeled as “avoidance” goals. Thus, with a *performance-approach goal* an individual aims to appear competent, whereas an individual with a *performance-avoidance goal* avoids appearing incompetent. Correspondingly, a *mastery-approach goal* refers to individual's desire to develop mastery over a subject while a *mastery-avoidance goal* arises out of a desire to avoid misunderstanding, deteriorating skills, or failure to accomplish everything that would be within one's reach with decent effort. In academic setting, a performance-avoid goal may result for example in test anxiety and avoidance of seeking help, whereas a mastery-avoidance goal may appear for instance in the form of poor study strategies and avoidant behavior. Both of them might manifest themselves in relatively lower achievement levels. As performance goals arise due to extrinsic motivation of some degree, the undesired manifestations of especially performance-avoidance goals resemble the ones pinned on extrinsic motivation above.

Interest. Finally, there are two kinds of interest [e.g., LP16, p. 93]—*individual* and *situational interest*. The former is caused by the already-familiar intrinsic motivation, whereas the latter is temporally short-lived and supported by prevailing context.

2.2 Contextualized Content

Without further definitions, the term *context* can be taken to refer to the set of all things in the universe, under direct or indirect influence of which some specific interesting phenomenon takes place [cf., CC10; PH14]. From this angle, it is crucial to specify, which kind of context some discussion refers to. In an educational setting, context could refer for example to

- culture, laws and statutes, rules of an educational institution
- large-scale geographical location: continent, country, municipality/city etc.
- small-scale geographical location: school, club, home, serene riverside, mass transport vehicle etc.
- field of study (some specific discipline, interdisciplinary)
- sociological setting: alone, with parents, with mentor, with friends etc.
- teaching and studying methods that are being used
- study materials and (technological) aids that are available
- real-world phenomena related to the concepts intended to be learned
- prevailing mental models (as in the schema theory), which are the base for constructing new knowledge (the constructivist perspective [BT11, p. 22])
- previous experiences that define learner’s prevailing world view, changing of which can be seen as learning (the phenomenographical perspective [BT11, pp. 22–23]), or
- neurological state of the learner’s brain, which changes along with learning (cf., constructivism and phenomenography).

In the following discussion, *context* refers to a well-defined real-world application area that is systematically being used as a basis for teaching in such a way that the actual concepts intended to be learned are derived from the objectives, needs, problems, phenomena and methods of it. For instance, Media Programming is an embodiment of such a context in the sense of presenting introductory programming concepts systematically through media-related examples. *Contextualization*, in turn, is the process of applying that kind of teaching method to a set of concepts to be taught.

Traditionally, university teaching has been *decontextualized* [CC10; GF05], meaning that the concepts intended to be learned have been introduced as themselves on theory level without context. This is the preferred way to have students to learn [GUZ10; TEW+08], the premise for which being that while contextual approach covers one application area, the actual concepts being taught are usually more general and intended to be applied to other application areas as well [CC10]. This intent is known as *transfer of learning* [GF05]—the ability to generalize and apply some knowledge valid in one context to other, equally valid, context. Unfortunately, this poses a problem, for it is discovered that students may learn to apply a concept only in the context it was initially learned [GF05; GUZ10].

On the other hand, students may not learn at all without some context [GF05]. In fact, Magana & Falk & Reese claim that practical contexts would actually benefit learners by helping them to associate problems with concepts and to transfer those concepts to other suitable contexts [MFR13]. This view obviously contradicts with the above-described issue with transfer of learning. From this viewpoint, it could be better to apply context, so that if the knowledge that students learn is not transferable, at least it is applicable in

that context. Any context that is used, must be relevant to the learners [GF05; TEW+08], and if the chosen context reflects the most common uses of that knowledge, at least the students would be able to utilize the knowledge most of the time.

In addition to the uncertainty related to the transfer of learning, there are some other considerations related to contextualization. For example, context cannot be expected to aid students to learn more content in less time—instead, the time necessary to internalize the content increases due to context [GUZ10] (see the next section). Also, some students may not “need” any context, and the contextualization may help only some portion of the students. At least in theory, this in turn raises an ethical question: Is contextualization fair for those students it does not help?² On the other hand, one can ask if it is fair that teachers can aid students in a way that in principle offers everyone the same possibilities, but choose not to do so. Is it not the purpose of teachers to maximize learning?

Contextualization has the potential to give motivation for learners [CC10], which of course is essential for learning [GUZ10]. At its best, context can accentuate the future usefulness and every-day value of the contextualized concepts [YG08; TEW+08], give the learners a feeling of authenticity [YG08; TEW+08; BT11, p. 37], appeal to learners’ personal interests [YG08; TEW+08], and foster creativity [CC10]. Especially the two latter ones also increase entertainment value of studying [YG08; TEW+08; BT11, p. 182] and encourage students to work harder [YG08; TEW+08]. The previous points can also lead to more effective lessons via focusing learners attention [KK13].

From theoretical perspective, context in general supports learner’s situational interest. Some contexts, such as Media Programming, are based on every-day phenomena, which, according to attribution theory, increases peoples’ motivation to understand them. The authenticity, every-day value, and future usefulness of a context affect several things:

- They increase the utility value of learning, whereas personal interests and creativity boost its intrinsic value. In accordance with expectancy-value theory, these two in turn increase individual’s anticipation while making the related costs to seem smaller.
- They—that is, relevance of course material—also support learner’s individual interest [BKK06, p. 479; PIN04] and might engage in transition from situational interest to individual.
- In the eyes of self-determination theory, the initial situation is at some level, but with the aid of context, the motivation level might increase, possibly sooner than without it, or at least might escape from decrease: Authenticity, every-day value, and future usefulness also feed individual’s need for competence, which in turn improves internal motivation.

Finally, the increase of individual interest—or intrinsic motivation—might result in (1) viewing the learning outcomes as mastery-approach goals over the other three options in achievement goal theory, and (2) a more complete transition from surface learning towards a deep-learning approach [BT11, pp. 24–27].

Regardless of attitudes towards contextualization, there is always a trade-off between concreteness and abstractness [YG08]: An excessive contextualization may distract students and obscure the concepts that should be learned [GUZ10; YG08], but dropping all context from teaching would forfeit all the possible benefits of context. A right balance is obviously essential [YG08], but how to find it? Guzdial states that an abstract concept

²The same ethical problem has been encountered for example in a study [AUV15, pp. 109–110], where all of the students observed did not benefit from the possibility of earning badges.

may be simpler than its contextualized version and thus easier to remember [GUZ10]. Also, Tew et al. claim that practical ideas have a larger probability of being remembered [TEW+08]. These two opinions clearly steer teachers towards trying to find the most simple way of expressing and illustrating content. Also, concepts and context must be clearly separated from each other [TEW+08].

2.3 Approaches for Contextualization and Multimedia Usage

Classifications are schemes designed to differentiate items of a large set (such as all living organisms) from each other and to aid comprehending the set as a whole. A classification can be a hierarchical system of classes, where the most common one contains and represents all of its subclasses, or a non-hierarchical one. It can also have several dimensions [for instance: MAN93] that could also be combined into a graph where each item in the set being classified inherits from all of the subgraphs—hierarchical or not—representing the dimensions. For example, in the field of computer science education research, one example of an extensive hierarchical classification is that of Kelleher & Pausch [KP05], which classifies programming tools designed for novice programmers. Their classification is based on (1) whether the tool actually teaches programming or just uses programming as an instrument to achieve other goals, and (2) in which way the tool is primarily designed to aid its users. On the other hand, as also Kelleher & Pausch, Guzdia [GUZ04] classifies alike tools on the basis of their influences on other tools by describing their relationships. Some more general examples of software development environment (SDE) classification criteria are presented in Table 2.1 below.

On the basis of literature review, it seems that currently there exist numerous small-scale programming tool comparisons [for example: UTT+10], but only a few of more generalized taxonomies or classifications related to them or the higher-level approaches they represent. Naturally, one reason for this is that while it is relatively trivial to compare and assess several mutually similar items, the full range of all available approaches and tools to aid novice programmers is so extensive that it becomes quite hard, or even impossible, to develop an iron-clad, unambiguous, and detailed classification for them. For one, there are certainly many ways to classify things, depending on which ones of their properties are of interest at a time. Which ones of those properties to choose

Table 2.1. Some examples of general software development environment (SDE) classification criteria (as described in [MAN93]).

Criterion	Classes
Nature of tool	SDE (for single language) Meta-SDE (for multiple languages) General toolkit Method-based environment
Degree of integration	Carrier \longleftrightarrow Lexical \longleftrightarrow Syntactic \longleftrightarrow Semantic
Targetted sociological scale	Individual \longleftrightarrow Family \longleftrightarrow City \longleftrightarrow State
Time of publication	For example intervals of five or ten years.

for a single-dimensional classification, or should one take them all to compose a multi-dimensional classification? Besides, depending on criteria, a single item could belong in several categories in one dimension. Is this acceptable, or should a classification be based on such characteristics that every item could be placed in one and only one category per dimension? What would these characteristics be, and how to define them unambiguously? For several dimensions, how to visualize them to fully comprehend and easily communicate the classification and its internal relations as a whole?

In order to answer to **RQ1**, this thesis differentiates contextualization approaches on the basis of a simple one-dimensional hierarchical scheme with several criteria. For the purposes of this classification, contextualization is defined broadly to include not only approaches where a general-purpose computer is used to write program code, but also all physical gadgets a teacher could use to illustrate programming concepts and/or ways and places to take advantage of programming. Of course, exactly what to include in the classification is more or less open to interpretations and debate, but since the scope of this thesis includes only a small portion of the full assortment of tools, namely, those related to media computation, the exact boundaries of this classification are not of the utmost importance here. Should a pen and a paper or a blackboard and a piece of chalk be included? Only if they themselves are being programmed or used to illustrate programming-related ideas essentially on the basis of their own unique characteristics specially related to computing—not just because they can be used as instruments to draw and to contain/transfer drawings about programming. The classification is visualized in Figure 2.1 on the next spread.

First, all approaches are divided up into three main branches on the basis of whether the context is achieved with software, hardware, or both. General-purpose computers used to write program code are of course hardware, but as they usually do not directly contribute to contextualization, the classification concerns itself only with the software being executed in them; this is why the word *essentially* is being used in the main-level branch titles. Only if an internal device of such a computer directly contributes to contextualization, should the approach making use of the part in question be considered to be part of this classification.

2.3.1 Solutions Essentially Combining Software and Hardware

The first one of the main branches of the classification presented in Figure 2.1 above is (1) *Solutions Essentially Combining Software and Hardware*, which contains both processor-based and processorless devices that are used with general-purpose computers. These (often external) devices are in turn divided up into two sub-branches on the basis of the existence of a processor which directly contributes to the context: If the device does not contain a processor that executes program developed by the user, the device belongs in the branch (1.1) *A Device Driven with a General-Purpose Computer* and is considered to be driven by a program that is executed by another processor. Otherwise, the user's program is executed by the processor of the device in question itself, and the device belongs under the branch (1.2) *A Device Programmed with a General-Purpose Computer*. Thus, the only thing that matters here is whether or not a processor is directly used to create context—processor's implementation approach and its existence (in case some processors exist but do not directly contribute to the context) are irrelevant. Finally, both of the two branches are further divided up into two sub-branches: One for internal (from the viewpoint of the general-purpose computer being used) devices (1.1.1 and 1.2.1), and one for external ones (1.1.2 and 1.2.2). A more detailed classification of these branches is out of the scope.

Examples of devices that belong in the former branch (1.1) include devices driven via serial, parallel, game, and USB ports [e.g., KB12]; musical instruments and sound recording/processing equipment driven via MIDI (*Musical Instrument Digital Interface*) [MID15]; as well as stage equipment, such as lightning, props, and pyrotechnics, driven via DMX (*Digital MultipleX*) interface [PLA15]. On the other hand, the latter branch (1.2) contains for example handheld devices such as mobile phones [e.g., RIL12] and PDAs (*Personal Digital Assistant*), media players, positioning system clients, and some gaming consoles such as Nintendo Game Boy Advance [TEW+08]; assembly kits such as Lego Mindstorms [LEG]; non-portable gaming consoles such as Microsoft's Xbox [MIC15C]; and development-focused technologies based on microcontrollers or digital logic chips and used via development boards, such as Arduino [e.g., KAF+14; KAF+13; LPD; ARD15], Raspberry PI [e.g., ABK15; RAS15], and PIC [MIC15A]. Some devices could belong under either one of the branches (1.1) and (1.2) on the basis of the way they work and are being used: For example, a robot [e.g., COW+12; HEL16] could be actively driven by software running on a general-purpose computer, or it could be programmable, so that a developer flashes a new software version into its memory, after which the robot works independently according to the flashed program.

2.3.2 Essentially Software-Based Solutions

The second main branch in Figure 2.1 is (2) *Essentially Software-Based Solutions*, in which the program code is being written and run on a general-purpose computer without any external devices contributing to the context. This branch has several sub-branches according to the primary focus of the contextualization approach being considered. In some cases drawing a line between two approaches can be difficult, or a tool (such as Kojo) may be deliberately designed to offer facilities for different approaches. Consequently, these contextualization categories are not mutually exclusive. Detailed classification of the other branches than (2.2) *Media Programming* is out of scope here, but some examples of software belonging (primarily) in them appear below:

- Branch (2.1) *Microworlds and Story-Telling* includes tools such as Alice [CAR16], Dragon Pathways [DPW], Kojo [KOG], Looking Glass [LGL], Magic Forest [MFO], Mama [MAM], Scratch [SCR], ScratchJr [SJR], and ToonTalk [TLK].
- Branch (2.3) *Games, Robots, Simulations, and Artificial Intelligence* contains for instance AgentSheets [AGE14B], AgentCubes [AGE14A], Clickteam Fusion [CL16], GameSalad [GAM14], Greenfoot [GFT], Jeroo [DOR14], Kodu Game Lab [KOD], Kojo [KOG], Robocode [RCD], Robot Virtual Worlds [RVW], Scirra Construct 2 [SCI16], SGP Baltie 3 and 4 [SGP16B; SGP16A], Squeak Etoys [ETY], Stencyl [STE16], and YoYo Games Gamemaker: Studio [YOY16].
- Branch (2.4) *Graphical User Interfaces* includes (2.4.1) *Generally-Available Third-Party Tools* as well as (2.4.2) *Course-Specific UI Templates*, of which the former includes for example the Java Power Tools library [RP08] as well as the ACM Java Task Force's graphical user interface library [ROB+08].

From the viewpoint of this thesis, the topic of Media Programming is naturally the most essential one. While it itself is defined and discussed in Chapter 3, in this classification it is represented by the branch (2.2), which also essentially answers to RQ2. The branch contains all the tools that (are primarily intended to) enable applying the Media Programming approach; some of the tools mentioned here are also discussed more in § 3.5.

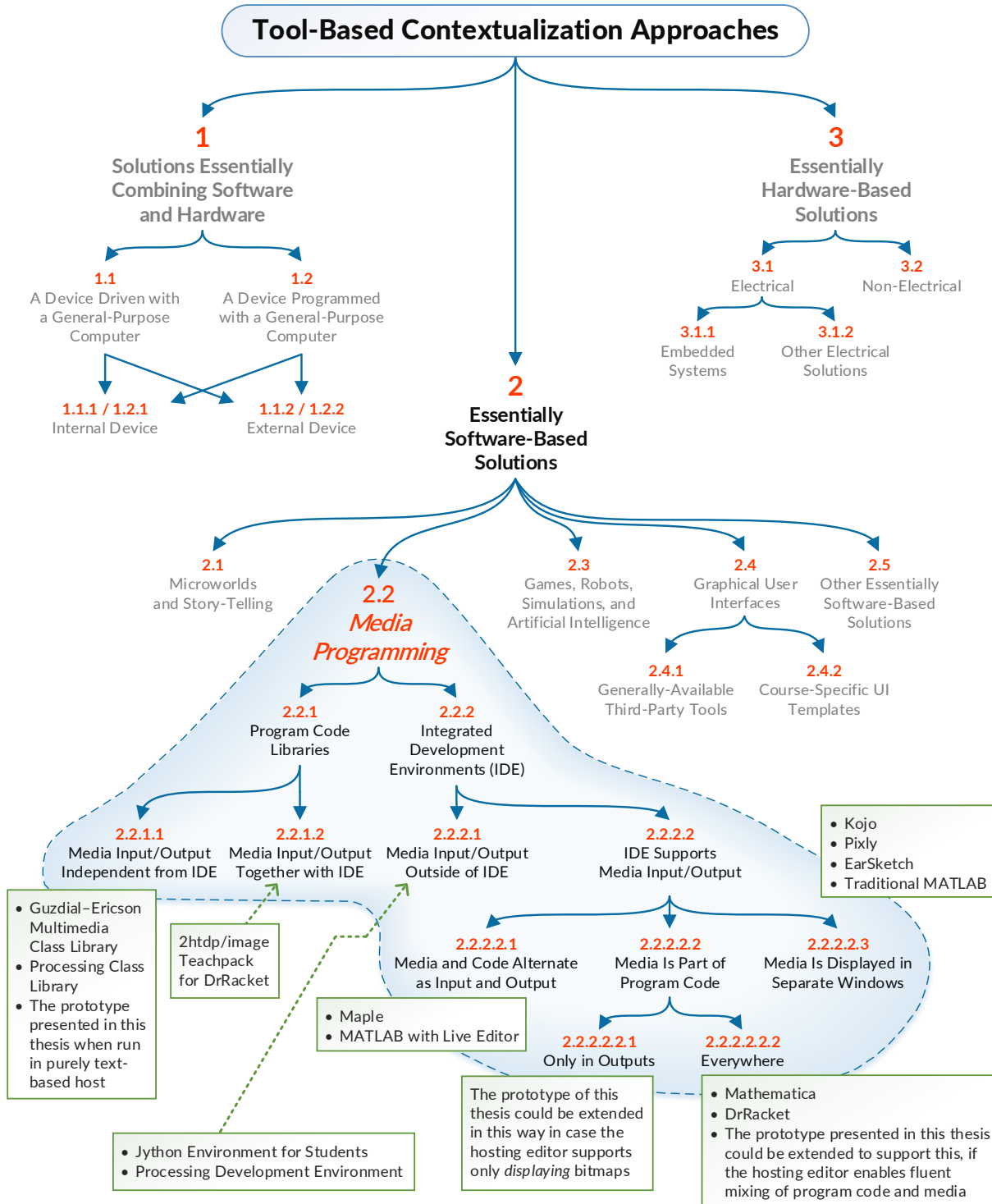


Figure 2.1. As an answer to **RQ1** and an partial answer to **RQ2**, this figure presents a classification of tool-based computer science contextualization approaches from the viewpoint of this thesis. The classification is basically a directed acyclic graph (DAG) that is denoted by the blue arrows. On many cases, however, the immediate sub-branches of a given branch are not necessarily mutually exclusive. The branch names excluded from the scope of this thesis are coloured as grey, whereas the branch that this thesis is concerned with, (2.2) *Media Programming*, is surrounded by the blue short-dashed line. The boxes next to the branch titles offer some examples of tools belonging in their related branches.

These tools are first divided into (2.2.1) *Program Code Libraries* and (2.2.2) *Integrated Development Environments (IDE)*, of which the latter includes both text editors, IDEs, worksheets, and REPLs. Data resource libraries, such as image and sound libraries, are not considered to be tools but merely data which can be processed with the actual tools.

The program code libraries—or for example *class libraries*—are further divided on the basis of the way they input and output (I/O) multimedia content in relation to the IDEs through which they are being used while developing code. Some libraries, such as the Guzdial–Ericson Multimedia Class Library and the Processing class library, belong in the branch (2.2.1.1) *Media Input/Output Independent from IDE*, the libraries of which do not use IDEs used to develop programs in performing their media I/O (operating system services, third-party libraries, and such are allowed for them to take advantage of). On the other hand, some IDEs offer solutions for performing media I/O, and the libraries—such as DrRacket’s [RCK] teachpack `2htdp/image`—that make use of those facilities belong under the branch (2.2.1.2) *Media Input/Output Together with IDE*.

The branch (2.2.2) is further divided up on the basis of the way the tools belonging under it utilize raw multimedia data in their user interfaces. This division criterion is independent from the appearance of program code: For example, the graphical code blocks of Pixly [PXY] (Figure 3.4), Scratch, and Snap! [SNP] are but program code—not multimedia. The branch (2.2.2.1) *Media Input/Output Outside of IDE* encompasses IDEs that do not participate in run-time multimedia input/output of the program being developed in any way. Examples of them include DrJava [DRJ]; Jython Environment for Students [JES] (Figure 3.1); Processing Development Environment [PRC]; Eclipse [ECL15]; NetBeans [NTB]; IntelliJ IDEA [JET15]; Microsoft’s Visual Studio [MIC15B], Windows Notepad, and MS-DOS Edit; Embarcadero’s RAD Studio, Delphi, and C++ Builder [EMB16]; MonoDevelop [MDv]; Visual Studio Code [MIC16]; Atom [ATM]; GNU Emacs [FRE15]; Notepad++ [HO16]; and vi. On the other hand, the user interfaces of the IDEs that belong in the branch (2.2.2.2) *IDE Supports Media Input/Output* offer varying levels of support for input and output of multimedia. This branch is still further divided up into three sub-branches.

IDEs in (2.2.2.2.1) *Media and Code Alternate as Input and Output* are able to display program code and multimedia in a single window without being able to mix them. Examples include MathWorks MATLAB’s [MAT15] Live Editor (Figure 3.6) and Maplesoft Maple [WAT15A] (Figure 3.7). On the other hand, (2.2.2.2.3) *Media Is Displayed in Separate Windows* includes IDEs offering support for displaying multimedia without integrating it into a single window together with program code. Examples of development tools falling into this category are Kojo (Figure 3.2) [KOG], Snap! [SNP] (Figure 3.3), Pixly [PXY] (Figure 3.4), EarSketch [FMV15] (Figure 3.5), and the traditional MATLAB (Figure 3.6), all of which display media resources in their own windows.

At this point it should be noted that the concept of *window* is quite an unclear one: Does it refer to a blank rectangle that is created by the operating system or windowing library that is responsible of the user interface, or is it a higher-level concept? Is it an application window or some “control” (such as a button, a scroll bar, a list box, a menu, etc.), the most of which are likely to consist of several low-level windows? What about (dockable) child windows, “frames”, and “panes”, or so called light-weight windows and controls drawn and managed by applications themselves in windows provided by operating systems and other windowing facilities? Or what if there is no centralized windowing system that takes care of the user interface (e.g., in some embedded systems and older applications for general-purpose computers)? Considering the contextualization approach classification, no implementation-dependent low-level details are significant, but even from

the user’s point of view, the matter is not clear at all. In case of two “application windows” it is easy to tell if two objects are inside of a single window or not, but that concept is too coarse-grained for the classification.

The next more fine-grained concept would probably be that of a *control*, but there is still the same problem of a missing water-tight, universal and unambiguous definition that tells exactly what a control is and what it is not. Even if that definition existed, it could still be too coarse-grained. What would be the next more fine-grained level and how to formulate an ironclad overarching definition for it from the viewpoint of a user—one that holds good for all possible (future) controls? Considering the aspects just discussed, this thesis does not even try to define the concept of window. Instead, the writer hopes that the examples given in § 3.5 clarify the writer’s views towards the classification.

Finally, IDEs in the branch (2.2.2.2.2) *Media Is Part of Program Code* are capable to mix code and media together in different degrees and is still divided up into two sub-branches: (2.2.2.2.2.1) *Only in Outputs* contains tools that are capable to display for instance list structures with multimedia elements inside of them, but are unable to accept the same kind of content as their input. On the other hand, the branch (2.2.2.2.2.2) *Everywhere* contains tools that can both input and output content with multimedia elements in it, as for example DrRacket (Figure 3.8) and Wolfram Mathematica [WOL15] (Figure 3.9) are able to do.

The prototype that is developed as a part of this thesis, has to be considered as two parts: (1) a Scala library that provides the actual Media Programming functionality and can be used practically from any (text-based) REPL running on a platform that provides Java-supported windowing and graphic capabilities, and (2) a front end supporting the library by displaying bitmaps. The library alone falls under the branch (2.2.1.1) *Media Input/Output Independent of IDE*, as it can be set to display all bitmaps in separate windows external to the REPL window. However, the library and the front end together work the way both Maple and MATLAB’s Live Editor do and end up into the branch (2.2.2.2.1) *Media and Code Alternate as Input and Output*. Furthermore, the prototype could be extended to offer functionality expected in both of the sub-branches of (2.2.2.2.2) *Media Is Part of Program Code*. Possibilities for these extensions depend on what is possible in the environment the library is being used from.

2.3.3 Essentially Hardware-Based Solutions

The last of the main branches, (3) *Essentially Hardware-Based Solutions*, is a relatively small one. It consists of contextualization solutions, in which software does not directly contribute to the contextualization, and is divided up into two sub-branches: (3.1) *Electrical* and (3.2) *Non-Electrical*. The former is divided into two sub-branches: (3.1.1) *Embedded Systems* and (3.1.2) *Other Electrical Solutions*. An example of an embedded system (3.1.1) is Cubetto [PRI16]. Non-electrical systems (3.2) include for instance board games, such as Robogem [PAA15], Code Monkey Island [COD14], and Robot Turtles [ROB14].

Chapter 3

Media Programming as a Context

After discussing motivation and contextualization in the second chapter, it is time to move on to the major theoretical topic of this thesis: The *Media Programming* approach. This chapter begins by discussing meaning of the word “media” and presenting three views to analyze it (§ 3.1). After those preliminary aspects, the chapter proceeds by discussing the definition of Media Programming (§ 3.2), during which it also answers to the [RQ3](#). The discussion continues by focusing on Media Programming’s effects (§ 3.3) as well as some aspects to be considered when applying the approach to a course (§ 3.4). The chapter concludes with a quick glance at some tools that currently support Media Programming (§ 3.5); this also concludes Chapter 2’s answer to the [RQ2](#).

The next chapter will shift the focus from theoretical background to the primary contribution of this thesis—a prototype which offers cognitive scaffolding for courses based on Scala and Media Programming.

3.1 Three Views into Media

Of the several meanings of the singular nouns *medium* and *media* (from Latin), this thesis is concerned with the one referring to *a method to transmit information*. The plurals of those words, *media* and *medias*, respectively, stand obviously for two or more of those channels. Several more or less abstract manifestations of them include the following:

- Mass communication: newspapers, magazines, radio and television broadcasts, movies, computer games and networks, posters, information screens etc.
- Written products: books, brochures, cards, signs, letters etc.
- Far-distance communication: telephones, facsimiles, mail, telegraphs, flag semaphores, signal lamps, smoke signals etc.
- Inter-personal near-distance behavior: speech, singing, facial expressions, gestures, touch, playing musical instrument etc.
- Senses: vision, hearing, touch, taste, smell, electroreception (sharks), magnetoreception (birds), infrared sensing (some snakes) etc.
- Data storage devices: mass memories, memory cards, diskettes, tapes etc.
- Natural phenomena: electricity, electromagnetic waves, mechanical vibration etc.
- Concrete matter: paper, air, water, copper, iron, stone, wood etc.

Mayer lists three views for analyzing media [MAY14A, p. 3]. The *delivery media view* is centered around concrete immediate end-user content-delivering mechanisms, such as

screens, projectors, speakers, phones, printed products, paintings, and unplugged voice. A more general interpretation could include also data storage devices, natural phenomena, and concrete matter. The *presentation modes view* focuses attention on content presentation methods, such as text, illustrations, animation, speech, singing, and playing musical instruments. Finally, the *sensory modalities view* is concerned with the senses transferring content to observers from their environments.

The term *multimedia*, which also refers to the usage of several of those media, may be familiar especially from its usage with computers, where it often means transmitting content by taking advantage of some combination of sound and images¹. Even if many people associate *multimedia* with that sort of usage, this definition is only a single one from a single viewpoint, and a narrow one at that. Furthermore, Mayer’s definition is even narrower, as it rejects the delivery media view while including only simultaneous presentation of words and pictures (including video); for example, music and other sounds than speech are excluded altogether. On the other hand, Friedland & Jain choose the sensory modalities view and define multimedia more formally and broadly to be two sets of synchronized data streams $\omega_1 \dots \omega_n$ and $\mu_1 \dots \mu_n$, where ω_k transmits observation data and μ_k metadata from sensor k , and the sensors themselves are for instance human senses or electrical devices that continuously make observations from the observer’s environment [FJ14]. *Multimedia computation* (not to be confused with Media Computation) in turn is defined as processing and analyzing that data.

3.2 Defining Media Programming

As previously mentioned, *Media Programming*² (*mediaohjelmointi* in Finnish) is a hyponym of Media Computation and is used to denote Media Computation with some constraints applied to its definition, as will be discussed below. *Media Computation* (*medialaskenta* in Finnish), in turn, is so-called data-first contextualization approach, in which students utilize general-purpose computer workstations to write simple programs to creatively produce and process (cf., *multimedia computation* above) media content, such as still images, sound waveforms, music, and videos.

The current mindset of Media Computation is primarily based on the presentation modes view. Different content delivery platforms are not of (primary) interest here, although the concept of Media Computation could obviously be easily extended in that dimension. The emphasis is not on the way the media is perceived, either, and currently only visual and auditory sensory systems are being utilized to perceive content in several presentation modes. However, it would not be impossible to extend the context to cover other senses than vision and hearing: Olfactory and gustatory³ as well as somatosensory (via haptic communication devices) sensory system data could also be included to the media content being handled. Nevertheless, these extensions are out of the scope here.

¹The primitive technical capabilities enabling playback and editing of simple and low-quality “multimedia” (still images, sound, and video) were a major marketing strategy of many computer manufacturers during 1990’s. At the time of writing, high-quality images, videos, and sound are taken practically as granted and are obviously manifested by the countless video files roaming around computer networks of the day; *multimedia* has ceased to be a buzz word.

²Freudenthal et al. use term *media programming* to refer Media Computation as it is; in this thesis, these two terms have their differences as described in this section.

³More about enhancing multimedia with olfaction and gustation information: [AK15; RAN+15; HM14; MUR+14; RAN+14A; RAN+14B; RAN+14C; AG13; RAN+13; RLD13; GA12; ISH+12; NAR+11; NM11; RAN+11; GA10; SUG+10; POR+09; RAM+07; BMM06; HYA06; YAN+03].

After several revisions [GUZ09A] over the years, Guzdial’s definition for Media Computation is as follows:

Media Computation (nicknamed “MediaComp”) is a contextualized approach to introducing computing using a ubiquitous theme of manipulating media. The critical characteristic of MediaComp is that students create expressive media by manipulating computational materials (like arrays and linked lists) at a lower-level of abstraction. Students manipulate images by changing pixels, create sounds by iterating over samples, render linked lists into music, and create artifacts like collages, music, and digital video special effects. In so doing, the students learn computation. [MCo.]

It is worth noting that while the spirit of Media Computation clearly appears to be introduction of students to programming (and to computer science through programming), the definition above does not mention *programming* itself at all. On the basis of this, one could ask for instance if media manipulation without constructing programs by writing code or combining blocks—for example via an application program that enables its users to work on low abstraction level to manipulate pixel values and sound samples as well as to graphically construct data structures—is still Media Computation.

Moreover, the definition above does not explicitly state, which specific actions are Media Computation and which are not. In practice, this may be impossible to define exactly, which may result in thin and vague lines to be drawn between individual perceptions. For instance, Guzdial & Ericson include turtle graphics⁴ in their Media Computing curriculum. However, one could argue that a single hairline between the two points drawn using a turtle does not contain much creativity or expressivity, which in turn could exclude it from Media Computation on the basis of the definition given above. A technical teaching viewpoint, on the other hand, speaks for inclusion of that line if presented through the context.

The term Media Programming is used to restrict the scope of this thesis to the very core of the current practice of Media Computation. This does not intend to suggest that the things falling outside of Media Programming may not be useful or included under Media Computation; it only aims to rule out the borderline and corner cases, which this thesis will not discuss. Thus, in this thesis, *Media Programming* refers to a contextualization approach that meets both the definition of Media Computation given above and the five criteria given below.

- I: Media processing has to be the primary focus of the context. Contextualization approaches that teach programming but focus primarily on other aspects than media processing—for the sake of the processing itself—are not Media Programming, regardless of their possible (programmable) utilization of multimedia assets. Some examples of such excluded approaches are microworlds, story telling, game development, robot and artificial intelligence programming, user interfaces, and all blends of those. Of course, sometimes it may be difficult to classify some materials or actions without doubt, and in fact, Guzdial & Ericson themselves use materials which clearly are focused on user interface construction, simulation, and game development instead of pure Media Computation [GE11, pp. 337–495].
- II: The programs created are composed of text or graphical blocks displaying elements of a textual programming language [in the sense of e.g., STE16; GOO15; PXY; SCR].

⁴For instance: [GE13, pp. 402–414; GE11, pp. 97–114, 201–230, 387–455; DCE10, pp. 68–89, 408–415; GE05, pp. 43–59, 373–379].

- III: The programs created are constructed directly by the learners themselves. For example, programs created by computers themselves on the basis of observing someone are not Media Programming.
- IV: The programs created are of algorithmic nature. Drawing anything using a single library function call is not Media Programming in itself, but by combining those calls in an algorithmic manner, the created program as a whole can become Media Programming. Also, drawing a hairline using drawing functions programmed by learners themselves would probably be included due to its algorithmic nature, but of course not many introductory programming courses present an algorithm for drawing an anti-aliased line on a graphic screen.
- V: The programs created are of creative nature [GUZ16]. The creativity of an exercise may be either real creativity of an individual student, or ostensible creativity included into the exercise by its creator so that the outcome of that exercise seems to be “creative”. For example, an exercise about programming a deterministic algorithm has technically no creativity in it. However, if the images produced by that algorithm can be taken as artistic and probably non-usual ones, the exercise will still fulfill the creativity criterion.

What comes to tool programs⁵ utilized when teaching and learning, the way the tools are being used is what counts—the tools are of no importance in themselves. Of course, some tools [e.g., HSC; KOG; SNP] can be used for both (limited) Media Programming and other kinds of contextualization approaches.

Furthermore, it should be emphasized here that although Media Programming as a subset of Media Computation could be interpreted to refer to the curriculum developed by Guzdial & Ericson, it does not refer to any single curriculum or a single set of examples and exercises defined by anyone. Instead, it is to be taken broadly to include basically anything that complies with the definition of Media Programming given above. Thus, if a course is based for instance only on turtle graphics, drawing with for example the Processing library, generating dynamic web pages, or giving examples about making conversions between media formats, it still is Media Programming as long as the above definition is fulfilled. Surely, these kinds of extreme examples may be unilateral or biased representations of Media Computation, but they still are instances of it.

By analyzing four Media Computation books at least co-authored by Guzdial or Ericson [GE13; GE11; DCE10; GE05], three main genres and several sub-genres of Media Programming can be identified. They are presented in Table 3.1 below, which answers to the **RQ3**. A notable thing about graphical media is that currently it deals only with bitmap graphic. No content based on vector graphic exists yet, although the curriculum would probably not be hard to expand in that direction at least to some extent. Also, Guzdial & Ericson include generation of dynamic web pages into their curriculum, but this aspect is not mentioned in the Guzdial’s definition above.

There are books [such as, MR14; MBK15] and other resources [e.g., PAR11] that could be taken to be utilizing Media Programming approach at least partly, and many others, which have some intriguing examples and exercises in the spirit of Media Programming but clearly are not designed to actually follow that approach [e.g., SW08; SWD15; BRO09; EFF00]. These books are not included into the analysis presented

⁵Some tools not (primarily) targeted for Media Programming: [CAR16; CLI16; SCI16; SGP16A; SGP16B; STE16; YOY16; RES15; AGE14A; AGE14B; DOR14; GAM14; DPW; ETY; GFT; KOD; LGL; MAM; MFO; RCD; RVW; SCR; SJR; TLK].

Table 3.1. As an answer to the **RQ3**, this table presents a classification of the current genres and exercise types of Media Programming [on the basis of GE13; GE11; DCE10; GE05].

Genre / Examples
1. Creating (Original) Information <ul style="list-style-type: none"> 1.1. Turtle Graphics 1.2. Sample-Based Sound Synthesis <ul style="list-style-type: none"> 1.2.1. Additive: Producing and mixing sine, square, and triangle waveforms 1.3. Note-Based Music Composing: MIDI music 1.4. Web Page Creation: Dynamically based on database content
2. Transforming Information <ul style="list-style-type: none"> 2.1. Filtering Information <ul style="list-style-type: none"> 2.1.1. Pixel-Based Bitmap Filtering; Uniform/Gradual, Entirely/Partly <ul style="list-style-type: none"> 2.1.1.1. Color Adjustments: Separating and exchanging color channels, color replacements, color gradients, brightness adjustments, sepia toning, negation, black-and-white conversion, red-eye reduction, and posterization 2.1.1.2. Geometrical Transformations <ul style="list-style-type: none"> 2.1.1.2.1. Affine Transformations: Rotating, scaling, flipping, and mirroring 2.1.1.3. Convolution-Based Effects: Blurring and sharpening 2.1.1.4. Miscellaneous Effects: Oil-paint 2.1.1.5. Edge detection 2.1.1.6. Steganography: Hiding information into a bitmap 2.1.1.7. Statistics: Finding most common color in an area of an image 2.1.2. Sample-Based Sound Waveform Filtering <ul style="list-style-type: none"> 2.1.2.1. Volume (i.e., Amplitude) Adjustments: Normalization, fade-ins and fade-outs, and distortion by intentionally inducing clipping 2.1.2.2. Pitch (i.e., Frequency) Adjustments 2.1.2.3. Structural Adjustments: Reversing, mirroring, and removing unnecessary parts 2.1.3. Frame-Based Video Filtering <ul style="list-style-type: none"> 2.1.3.1. Time-Parameterized Bitmap Filters 2.2. Compositing Information <ul style="list-style-type: none"> 2.2.1. Primitive Shapes → More Complex Shapes 2.2.2. Bitmaps → Visual Collages; possibly with transparency and chroma-keying 2.2.3. Dynamically Changing Bitmap Collages → Videos 2.2.4. Sound Waveforms → Audio Collages 2.2.5. MIDI Clips → Audio Collages
3. Transforming Representation <ul style="list-style-type: none"> 3.1. Writing image's color components as values into a text file, modifying them in a spreadsheet application, and reading them back into an image 3.2. Writing samples of a sound waveform into a text file and representing those values as a line chart in a spreadsheet application 3.3. Interpreting sound waveform's sample values as colors to create a bitmap representation of the sounds 3.4. Retrieving a web page in source code (HTML) form and parsing some data out of it

in Table 3.1. There are also reports of several variations of Media Computation [e.g., FMV15; FRE+14; RDW13; MAG+13; GKX12; FRE+10; CP05], but those are excluded from the analysis, too.

3.3 Effects of Media Programming

As there obviously is no research data available concerning Media Programming as defined in this thesis, this section discusses results achieved with the more general approach—Media Computation—with the expectation that those results would be applicable to the constrained form of Media Programming defined in the previous section.

Media Computation was developed [GUZ13] in the Georgia Institute of Technology, where from the Fall 1999 onwards, every undergraduate student was required to take a course in computer science. The pass rates at the time for that introductory course (so-called CS1) were acceptable (about 78 %) for the whole university, but for the minority of students not belonging to the disciplines of engineering and computing but having a major related for instance to art, architecture, or business, the pass rates were regularly below 50 percent. Also, the majority of these non-technical students were females, and the portion of them passing the course was usually lower than that of males.

The original developers of the Media Computation approach hypothesized [GUZ13] that increasing attractiveness and relevance of the introductory CS course would

1. reduce plagiarism
2. decrease the portion of students failing the course or dropping out from it
3. induce more female students to participate and work more to increase the portion of female students that pass the course
4. students participating in the contextualized course should learn as much as the students on corresponding decontextualized courses, and
5. after the contextualized course, students should have more positive view of computer science and they should preferably be more interested about it.

During the past over 10 years, a considerable amount of evidence was found for two of the hypotheses described above. First, at Georgia Institute of Technology, success rates⁶ for the contextualized course instances were clearly higher than before starting to apply the Media Computation approach: The average failure rate of non-engineering students dropped from over 50 percent to below 15 percent and has stayed there since. Second, both the possibility for being creative and the feeling of course content’s relevance encouraged female students to engage more to the course content, which resulted in quite similar success rates between males and females, or even better ones for females than males. Congruent results have been obtained also in some other educational institutions for both majors and minors [e.g., LEE13; SIM+10; ST08].

As Media Computation is a method for contextualizing teaching, naturally everything previously discussed about contextualization-related motivation (§ 2.2) is applicable also to it. The creativity aspect, which the quoted definition in the previous section describes as an central part of Media Computation and which has earned Media Computing a sobriquet “Computing as Creation” [DEN14], deserves a few more words. While Media Computation is not about teaching art, it encourages students to be creative, to experiment, and to express themselves—to employ their own visions and artistic ideas.

⁶Success and failure are defined in accordance to a grading system that is commonly used in the United States of America: Success is defined as getting a passing grade of A, B, or C, whereas the lower grades D and F as well as withdrawals from courses are taken as failures [GUZ13].

From a motivational perspective, experimentation gives some freedom to learning-related inquiry, and that—together with creativity and artistry—increases individual’s feel for autonomy [SAW06, pp. 477, 480]. Except that it may further transition from situational to individual interest [CA16, p. 93], according to self-determination theory, it helps to increase learner’s intrinsic motivation⁷ [e.g., MA11, p. 225; PIN04]. Creativity is also in the core of the Maslow’s need hierarchy’s fifth level—the self-actualization.

3.4 Applying Media Programming

As is known for example from the Rainfall Problem [e.g., GUZ15, pp. 21–23, 27; SEP+15], learning the basics of programming itself can be a challenging task: While trying to solve computational problems and to learn computational/algorithmic thinking, the learners have both to master new ways of thinking and to remember at least a considerable amount of small details of programming-related concepts, programming language syntax, class libraries, development environments, and other tools that are being used. On top of all that, there is the problem itself as well as aspects related to a specific solution—for example program structure as well as variables and their roles [MMG15]. Consequently, the learners should not be burdened with technical aspects related to any context that the teaching is placed in.

To facilitate learning when applying Media Programming to a curriculum, students need what is known as *cognitive scaffolding* [e.g., PRI14, pp. 26–27; SAW06, pp. 11–12]—measures taken towards reducing the *cognitive load* [e.g., MMG15] falling upon the learners. In this case, one way to offer such a scaffolding is to provide a working environment that hides unnecessary technical details as much as possible and thus enables for the learners to concentrate on what really is essential considering the concepts that should be learned. This leads to the main contribution of this thesis—a prototype that offers such a scaffolding for Scala-based introductory programming teaching. The prototype is discussed in the following chapters.

Media Programming offers a good possibility to improve teaching, but some points need to be taken into consideration while preparing a course based on it. Whereas technical facilities enabling Multimedia Computation also make it possible for educators to create for example audiovisual illustrations⁸ and exercises to help understanding, for some people, it also poses problems. For example, while one of the core purposes of Media Programming is to *encourage* students to be creative and to *offer possibilities* for self-expression, the assignments given to the students must not *require* artistic talent. All exercises must be passable also for people with no previous exposure to the context provided, be it creating for example graphical or musical art.

Various inabilities will also provide challenges for some learners: For instance, students who have some level of color-blindness cannot be expected to complete any assignment that is based on identifying or using specific colors. In the same way, people suffering from amusia (tone-deafness) [e.g., PH03] may have hard time for example in recognizing specific notes, intervals, scales, melodies etc. by hearing. Furthermore, blind or deaf people will obviously not be able to complete any assignment that would require them

⁷A research also indicates that once external stressors, such as money, were removed, autonomy and work satisfaction correlated strongly in a sample of 1000 employees of a large company [e.g., MA11, p. 226]. Pink calls them as *threshold motivators* [PIN04], which correspond roughly the two—or maybe three—lowest levels in the Maslow’s need hierarchy.

⁸These could be discussed for example in the light of the *Multimedia Principle* [MAY14B, pp. 174–205], but that is outside the scope of this thesis.

to see or hear something, respectively. Ordinary text that is composed of for instance Latin alphabets and Arabic numbers—and even musical notation to some extent—can be passed to a blind learner for example in speech or in Braille code, possibly with the aid of a screen reader and a refreshable Braille display. However, no reasonable bitmap image can be understood by a blind person without someone creating an approximate textual description of it, even if the color component values of individual pixels were somehow passed to them. Screen readers cannot help them in this aspect, either⁹. Fortunately, as long as a student is not both blind *and* deaf, they are able to do exercises designed for the working sense [e.g., GUZ09B]: A blind student can work with sound, and vice versa.

The creativity dimension of Media Programming can be boosted by running (Internet-based) galleries and art shows exhibiting works created by students [PS13B; GUZ13; GUZ16]. For example, some exercises could be submitted by uploading the solutions to a gallery site accessible to all students participating in the course, and the best works could be exhibited as printed copies in the school premises. This in itself may improve students' feel of relatedness (Maslow's third level) and induce esteem from their peers (Maslow's fourth level). Relatedness, in turn, is a factor that improves intrinsic motivation, as described by self-determination theory [e.g., MA11, p. 225]. In addition, positive input from peers and instructors increase learner's individual interest [SAW06, p. 484; CA16, p. 93]. What is more, the feeling of empowerment induced by the observation of being able to express one's artistic visions may qualify as a direct experience of mastery, which in turn should improve learner's self-efficacy [e.g., MA11, p. 225]. This may result in learner's choice to continue studying and not drop out, which for one may explain the success of Media Computation in improving success rates (see § 3.3 above).

"Gallerization" could also work as a form of gamification: Competitive students could be inspired to do more work to produce the most imposing work in the gallery and possibly to receive most votes and badges (a performance-approach goal from achievement-goal theory). This could also be considered from behavioristic viewpoint: Positive peer input and esteem could work as reinforcements that cause students to become conditioned to them and work more to keep them coming and to increase their amount.

Integrating galleries to Media Programming course also has possible complications, however: Noncompetitive students may not be inspired by the approach (an ethical problem of equality), timorous students may become distressed and evasive by the (forced) publishing-competitive aspect, and the interaction between an over-competitive student and the other students may get unpleasant. What is more, in terms of achievement-goal theory, the creative aspect may cause some people to actively avoid of being seen as untalented, which causes them to see the corresponding tasks as performance-avoidance goals. Finally, students should have the option of removing their work from any and all galleries after the course, and preferably not to make them public in the first place, if they so wish.

An example of a milder approach to foster creativity is that of producing personal portfolios [GKX12]. It avoids the possibility of public shame and injurious competition between students, but might still be motivating enough¹⁰ at least for visually oriented students to work more to get a better-looking portfolio, even if the portfolio itself would not be assessed in terms of artistic talent and vision.

⁹While machine learning could be utilized to enable screen readers to identify single objects from images on the screen with some accuracy, these applications are far from infallible: Even in the best circumstances, they can recognize only that limited number of items they have been "taught" to recognize, and first and foremost, they do not think or understand what they see. They can offer suggestions, but they cannot be trusted.

¹⁰This situation could be comparable to a research concerning the usage of achievement badges [AUV15, pp. 102–103], in which the badges clearly helped (some) students to work more, even if the badges they received were not visible to other students.

3.5 Tools Enabling Media Programming

The reason for both the topic and the content of this thesis to be such as they are now, is the lack of integrated development environments (IDEs) and libraries that both enable Scala-based development and offer students cognitive scaffolding for Multimedia Computation to succeed. However, for other programming languages, there are several tools which support Media Programming with more or less scaffolding. The purpose of this section is not to extensively assess these tools or for instance any results achieved with them. Instead, this section complements the answer to the [RQ2](#) (§ 2.3.2) by briefly introducing the tools and their essential characteristics from the viewpoint of this thesis with focus on bitmap processing and usage of programmatically-created media. The discussion begins with IDEs and ends with a short glimpse of a few code libraries.

The current selection of IDEs includes a bunch of candidates among of which to choose a tool for a course based on Media Programming. The main tools that this subsection is concerned with are Jython Environment for Students, Kojo, Snap!, Pixly, EarSketch, DrRacket, MATLAB, Maple, and Mathematica. More traditional IDEs are not discussed here, but some of the Scala-related ones are briefly presented in the § 4.3. The program code used in the figures in this section is presented in the Appendix B.

All of the nine IDEs just mentioned enable development of programs expressed in either textual programming language or graphical blocks standing for elements of a textual language. The first six of them are intended for introductory programming and provide cognitive scaffolding for instance by

- offering simplified programming APIs,
- offering beginner-friendly language variations, including blocks that inherently enforce syntax rules of the programming language being used, or
- shielding the learner from the complexities of professional IDEs used daily in software development.

The three latter of the IDEs just mentioned are primarily intended for mathematical applications instead of professional software development, although all of them are (and have to be) more or less applicable for professional programming. They also offer interfaces for external usage from other programming languages and IDEs.

Jython Environment for Students. The first IDE presented here is the Jython Environment for Students [JES]. It is a simple development environment aimed for studying the Java implementation of Python—Jython [JHN]—according to the Media Computation approach, and has been developed as an open-source project under supervision of Mark Guzdial. The user interface of JES is quite simplistic: As can be seen from Figure 3.1, the main window has been divided into three main components: a traditional text editor, a REPL below it, and a side pane displaying for instance function descriptions. As a slightly advanced functionality, JES also makes it possible to watch values of variables. The REPL in turn serves as a console for any textual output and errors produced by the program that is being run.

As the actual IDE part of JES does not participate in presentation of the media results produced at run-time by the programs written in the text editor of JES, in the classification presented in § 2.3 JES belongs in the branch (2.2.2.1) *Media Input/Output Outside of IDE*. The three windows displaying processed images are created run-time by the code written in the text editor of JES and are provided by a separate application (PictureExplorer) bundled with JES. In addition to displaying bitmaps, they provide a simple zooming functionality as well as a possibility to inspect the color component values

of individual pixels of the bitmap that is being displayed. There are similar programs for sound waveforms and videos, but they are not discussed further in this thesis.

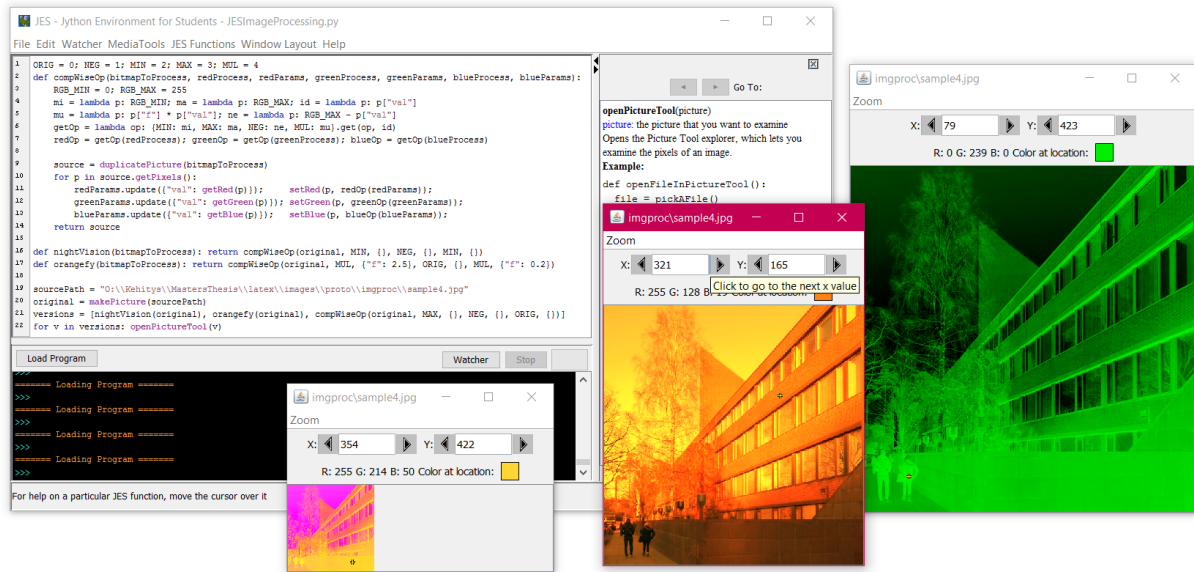


Figure 3.1. Trivial bitmap manipulation in Jython Environment for Students [JES] version 5.01. The program code used in this figure is better presented in Code Example B.1.

Kojo. In the classification of § 2.3, the next focus of discussion are the sub-branches of (2.2.2.2) *IDE Supports Media Input/Output*. The first branch of them to be discussed is the (2.2.2.2.3) *Media Is Displayed in Separate Windows*. The first IDE to be presented from this branch is an open-source project called Kojo [KOG], mainly developed by Lalit Pant of an Indian-based Kogics Foundation. As can be seen from Figure 3.2 below, the main window of Kojo has several child windows. At the top right there is the graphical canvas, which displays the turtle graphics by the program in the editor window below. At the right side of the editor, there is an output window for textual output and error messages. To the left of the canvas, there is a window displaying a full call-by-call program execution trace (not the call stack) when the program is executed with the tracing feature on. Kojo also takes advantage of its built-in story-telling functionality by displaying palettes (such as the one in the leftmost child window). They serve as simple reminders of the library API, in addition to which each of the items in a palette call can be inserted into the code editor by clicking them with a mouse.

Kojo is a project that combines several other open-source components into a single IDE. At the moment of writing, it is also the only IDE that offers some level of support for Scala-based Media Programming approach: In addition to the programming language being Scala, Kojo provides library support for turtle graphics and trivial note-based MIDI music composing. Kojo also provides some support for other contextualization approaches, such as story telling, mathematics, simulations, and even Arduino-based embedded system development. However, especially both pixel-based bitmap processing and sample-based sound processing are unsupported at the moment, and the usable feature set from the viewpoint of Media Programming is that of turtle graphics. Also, while four short introductory volumes have been written about Kojo, a comprehensive documentation concerning library APIs and the IDE itself seems to be missing at the moment.

Despite the shortcomings listed above, Kojo has several very useful features. For starters, it has a built-in worksheet-style code editor with syntax coloring and basic code

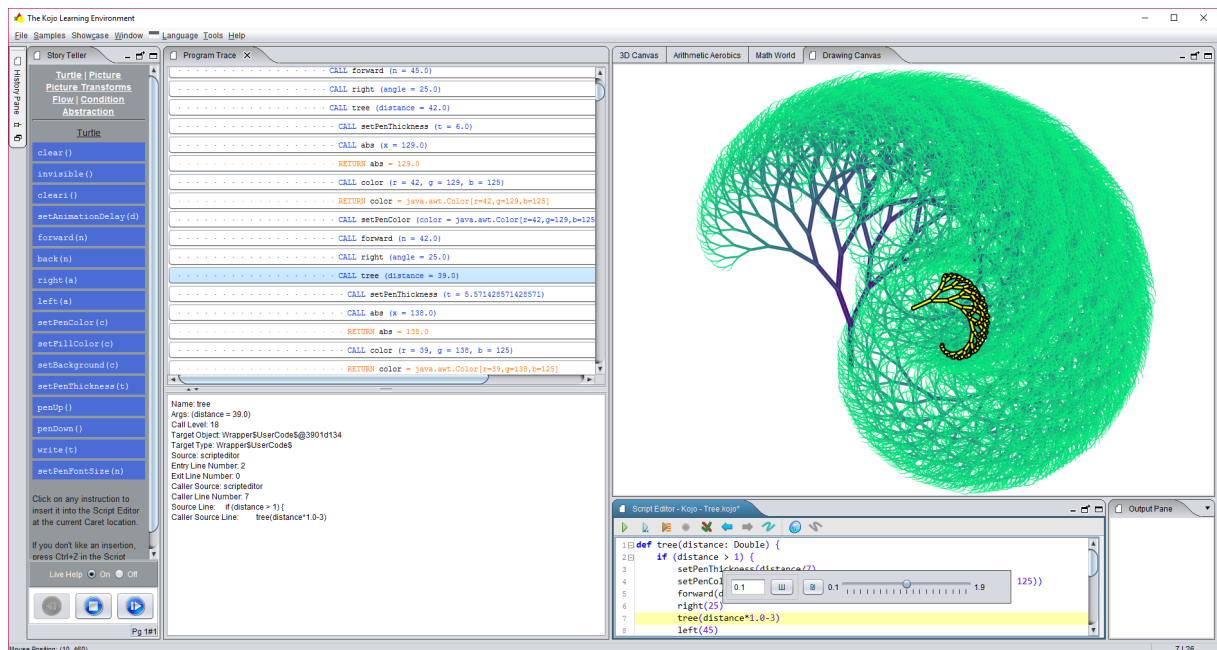


Figure 3.2. Kojo [KOG] version 2.4.06 r41 application window containing a simple recursive tree and the program that draws it. The program code is available in Code Example B.2.

completion. The code completion can even be set to include only the content appropriate for the context (i.e., turtle, staging, math, 3D) that is being used. In addition, the code editor displays pop-up sliders¹¹ for adjusting values of numeric literals; such a slider is shown in Figure 3.2 overlaid on top of the code editor. The tracing execution mode in turn is aware of Kojo’s own turtle API and is able to highlight the effects of drawing operations either with a bounding box that limits the affected area or by highlighting the actual drawing operations, as is done in Figure 3.2; this is especially useful when studying recursive algorithms.

Snap!. A second example of IDEs that belong to the branch (2.2.2.2.3) is Snap! [SNP], the user interface of which is presented in Figure 3.3 on the following page. Snap! is an extended open-source reimplementaion of Scratch [SCR], written by Jens Mönig and Brian Harvey. As such, development is based on the notion of a *sprite*—that is, a character—to which developers can attach program code (*scripts*), bitmaps (*costumes*, such as the dog in the figure), and sounds. The sprites can react to several things, such as the press of the run button (the green flag in the upper-right corner), key presses, mouse events, sprites touching each other, and broadcast messages sent by other sprites. In addition to the sprites, there is a stage (the large white area in the figure) in which the sprites are displayed. The stage can also have program code attached into it, but the possible functionality is more limited than that available for sprites; after all, actions like moving the stage to a different coordinates do not make much sense.

All program code in Snap! is presented as graphical blocks, using which the programmers literally builds their programs. As Figure 3.3 illustrates, on the left edge of Snap!’s user interface there is an area from which these blocks can be drawn into both the scripts area and the block editor. In addition to for instance mathematical expressions, control structures and variable management (including list handling), there are blocks for movement, pen control (the sprites can act as turtles in the sense of turtle graphics), manipulating costumes, asking input from users, and playing sounds. In addition,

¹¹These sliders are familiar for instance from the JavaScript code editor used in Khan Academy.

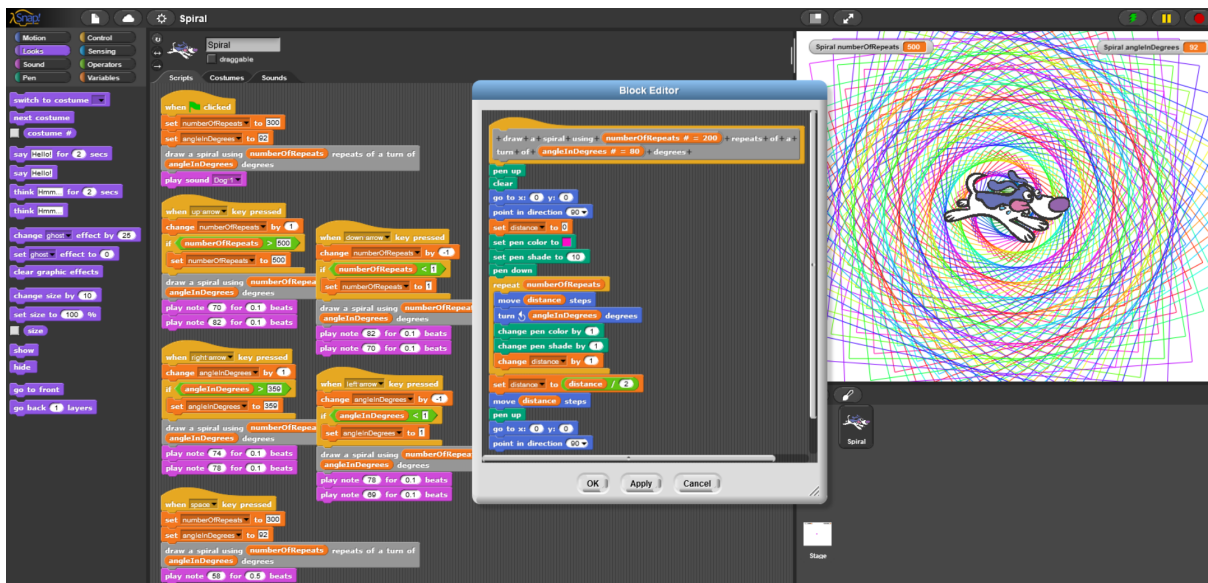


Figure 3.3. The user interface of Snap! [SNP] version 4.0.6 in a web browser. In it there is a simple program for drawing some trivial turtle graphics and playing sounds. The program code is better presented in Code Example B.3.

programmers can use these built-in blocks to define their own blocks that act as procedures (*commands*), functions (*reporters*), or conditions (*predicates*). An example of such an definition can be seen in Figure 3.3, in which a parameterized procedure for drawing spiral-like images is defined in the block editor.

Snap! itself fits for contexts such as story telling and turtle graphics, of which the latter is the reason for Media Programming practitioners to look for it. For more extensive functionality, there exist both extension plug-ins and extended versions of the whole Snap! environment that provide some support for their specific contexts. Some examples of those contexts are Arduino [ARD15], physical controllers such as Nintendo’s Wii Remote and the Leap Motion [LEA16], several robots such as Lego Mindstorms [LEG] and Parallax Scribbler 2 [PAR16], generative art, agent-based simulations, and speech synthesis. As with Kojo, however, for instance both pixel-based bitmap processing and sample-based sound processing are unsupported at the moment, although someone could of course write an extension to enable those application areas.

Pixly. From the viewpoint of Media Programming, both Kojo and Snap! offer little more than turtle graphics. Fortunately, there are other possibilities. As a third example of IDEs that belong in the branch (2.2.2.2.3), this thesis presents the Pixly [PXY] (Figure 3.4). It is a browser-based application for performing one of the core areas of Media Programming—the pixel-based bitmap processing—and is written by Jake Trower and Katherine Hill under supervision of professor Jeff Gray at the University of Alabama. The blockish program code is similar to that of Snap!’s, so it will not be discussed here further. The important thing here is that Pixly provides blocks for getting and setting pixels and their color components, which enables the pixel-based processing in the first place. Pixly also has a sibling called Tunely [TNY]; it is not discussed here, but a trivial code example for it is presented in Code Example B.5.

EarSketch. The fourth example from the branch (2.2.2.2.3) *Media Is Displayed in Separate Windows* is an application called EarSketch [FMV15]—a contextualized browser-based IDE for introductory programming courses. The context itself is quite an interesting one—loop-based music sequencing. The application is developed by a team having members mainly from several North-American educational institutions, such as the Georgia

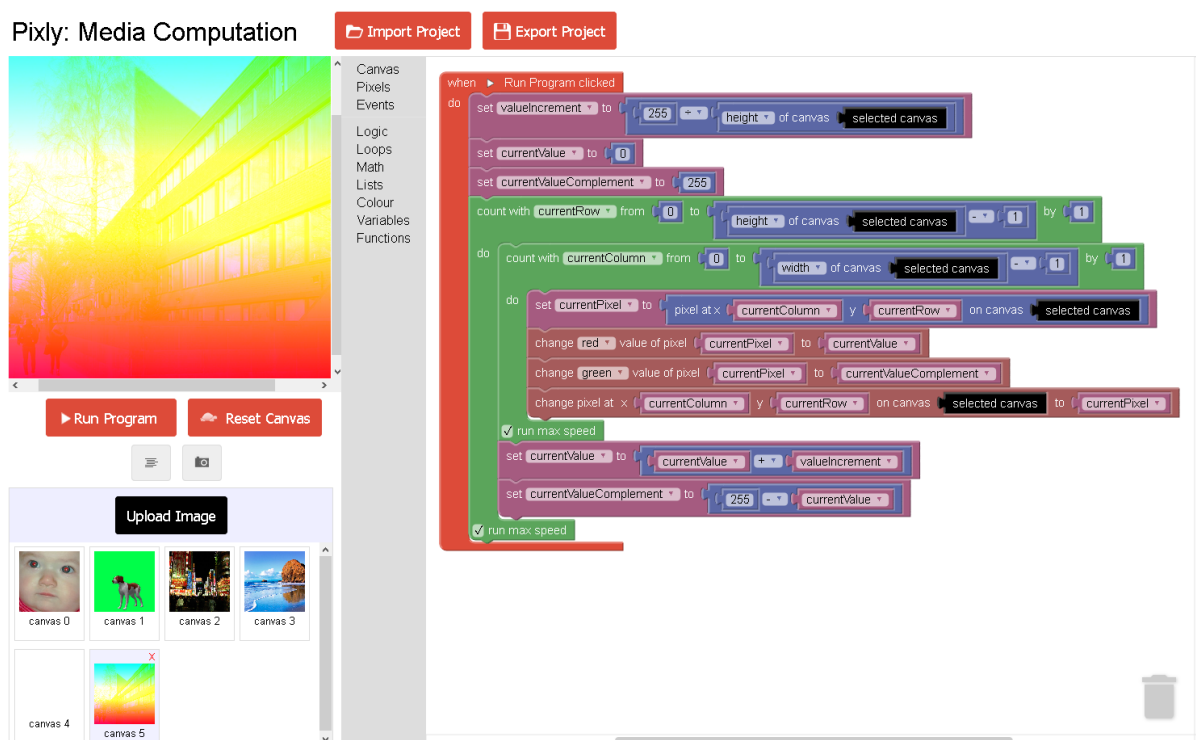


Figure 3.4. Trivial bitmap manipulation in Pixly [PXY]. The program code is better presented in Code Example B.4.

Institute of Technology, in which also the Media Computation approach was developed. The programming languages supported by EarSketch are Python and JavaScript, with which the student is expected to write programs that generate mainly clip-based sound content and automation, such as volume, panning, and effect parameter curves, for audio tracks visible in an audio sequencer track view. The practical workflow is to write a program, then actually generate the content and automation by running it, and finally play the content if it looks about right on the tracks.

As can be seen from Figure 3.5, the user interface of EarSketch resembles other IDEs quite much despite its musical context. The main parts of the IDE are in the middle column: The code editor is at the bottom and the audio sequencer view containing audio tracks—the principal “media” of EarSketch—on top of it. The audio sequencer view can also be replaced with canvas for exercises that require drawing. The views on the side can hold several kinds of content, such as lists of sound loops available (on the left) and the EarSketch-based curriculum (on the right).

The audio sequencer view of EarSketch has the most essential functionality of any professional sequencer. Audio tracks display their audio clips with waveforms, whereas automation tracks display automation curves for effect parameters. Each audio track can be individually soloed and muted, whereas automation tracks can be hidden globally and bypassed on track-by-track basis. The tracks can also be zoomed both horizontally and vertically. The play head position can be set by clicking the track view, and the content can also be set to be played in a loop mode. Even a simple metronome can be set to click during playing. The available effects are simple, but for an absolute beginner, the selection is probably enough to play with. In addition to volume and panning, which could be considered as basic parameters of each audio track rather than separate effects, EarSketch offers such filters as three-band equalizer, band-pass and low-pass, compressor, reverb, chorus, flanger, phaser, ring modulation, distortion, tremolo, and wah-wah.

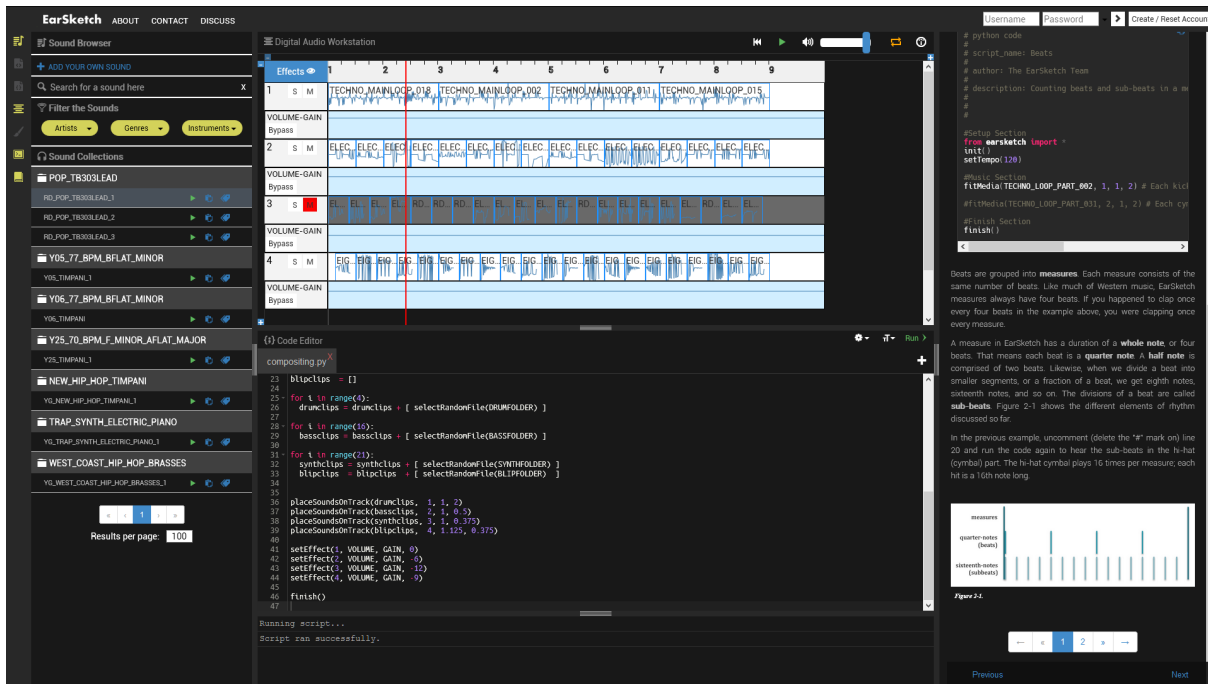


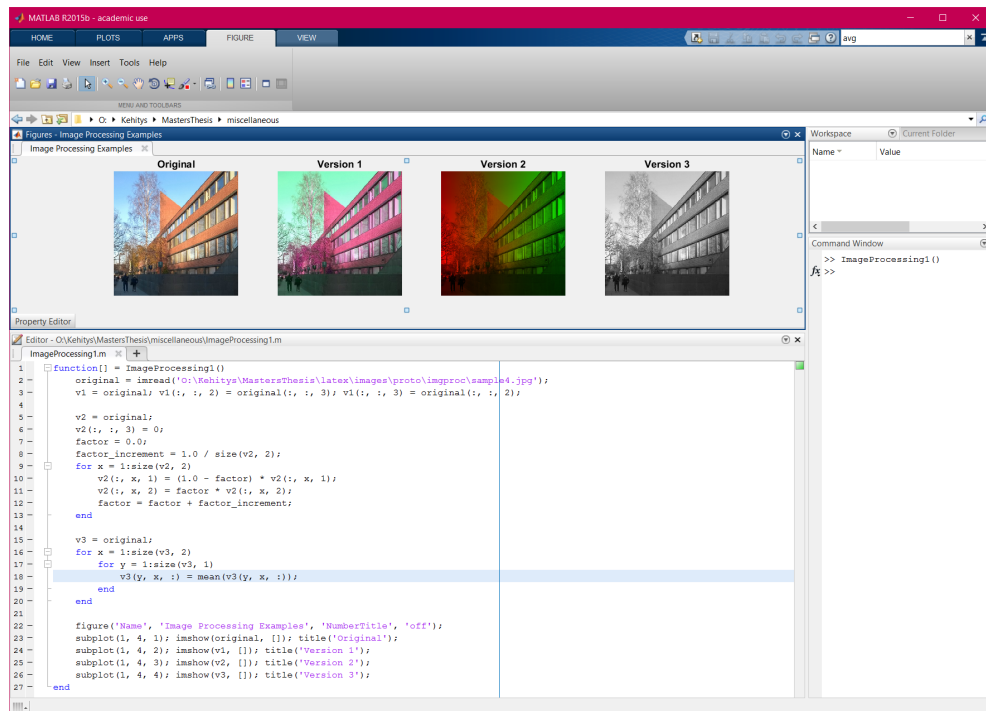
Figure 3.5. Clip-based sound waveform compositing in EarSketch [FMV15]. The program code used in this figure is presented in Code Example B.6.

MATLAB. The final example from the branch (2.2.2.2.3) is a general-purpose commercial platform for developing applications which perform numerical calculations and create illustrations: MathWorks MATLAB [MAT15]. This thesis distinguishes two different flavors of programming with MATLAB: a so-called traditional approach (Figure 3.6a) as well as the Live Editor approach (Figure 3.6b). The latter facility is included into a new version of MATLAB that was published during the writing process of this thesis.

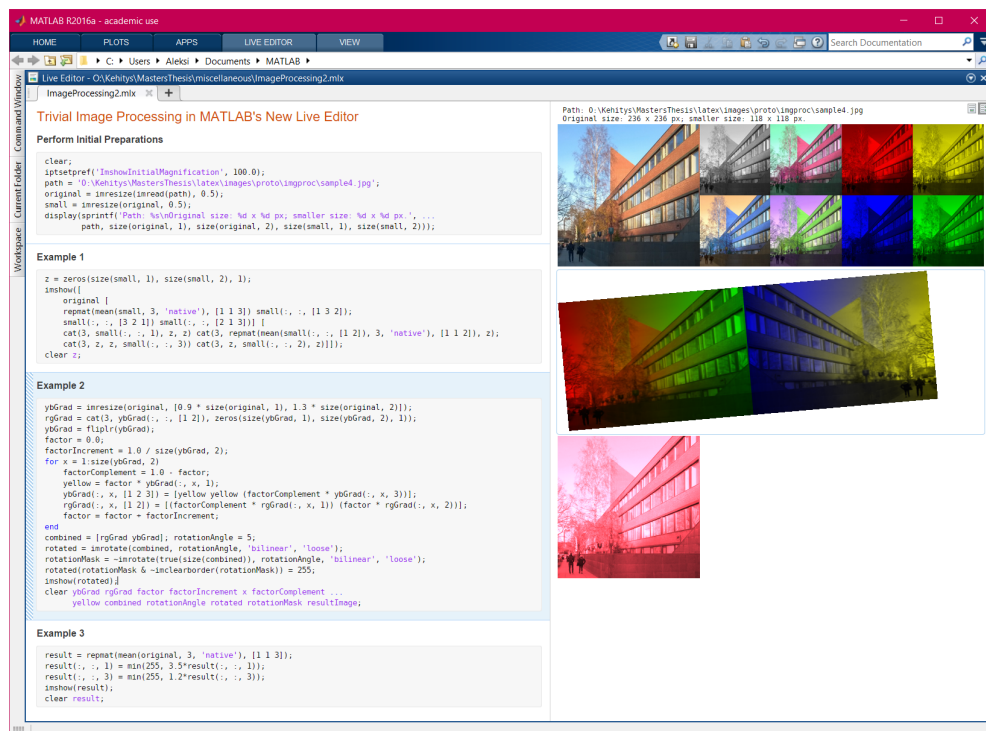
From the viewpoint of the contextualization approach classification, MATLAB is as a cross between the branch (2.2.2.2.3) and the branch (2.2.2.2.1) *Media and Code Alternate as Input and Output*. The traditional approach mentioned above is equivalent to a simple text editor while the media is displayed in separate (dockable) windows; in this case, MATLAB belongs under the branch (2.2.2.2.3). On the other hand, the Live Editor enables programmers to make use of a capability of alternation between explanatory content and executable code fragments to create textbook-style material. The execution results of the code fragments can be displayed either beneath them, for instance as in Figure 3.9, or next to them in a separate column, as presented in Figure 3.6b. Thus, as the Live Editor enables alteration of program code and media in a single window, is MATLAB in that case placed into the branch (2.2.2.2.1).

In addition for instance to Jython Environment for Students, Processing, and the combination of DrJava and the Guzdial–Ericson Multimedia Class Library, MATLAB is one of the tools that are known to have been used for Media Computation [LH10]. For professional bitmap processing and analysis tasks, MATLAB’s Image Processing Toolbox offers an extensive set of functionality. What is more, some of that functionality can take advantage of graphic processing units (GPUs), and for a subset of the functionality, stand-alone program code in C language can be generated.

A problem with MATLAB is that both it and the Image Processing Toolbox are made to be professional workhorses—not to be easy to use for a novice programmer or an artist. The question is not *if* something can be done, but how deep knowledge of mathematics,



(a) A traditional script in MATLAB R2015b (8.6.0). The program code in this figure is better presented in Code Example B.7.



(b) A live script in the Live Editor of MATLAB R2016a (9.0.0). The program code in this figure is better presented in Code Example B.8.

Figure 3.6. Trivial bitmap manipulation in MathWorks MATLAB [Mat15].

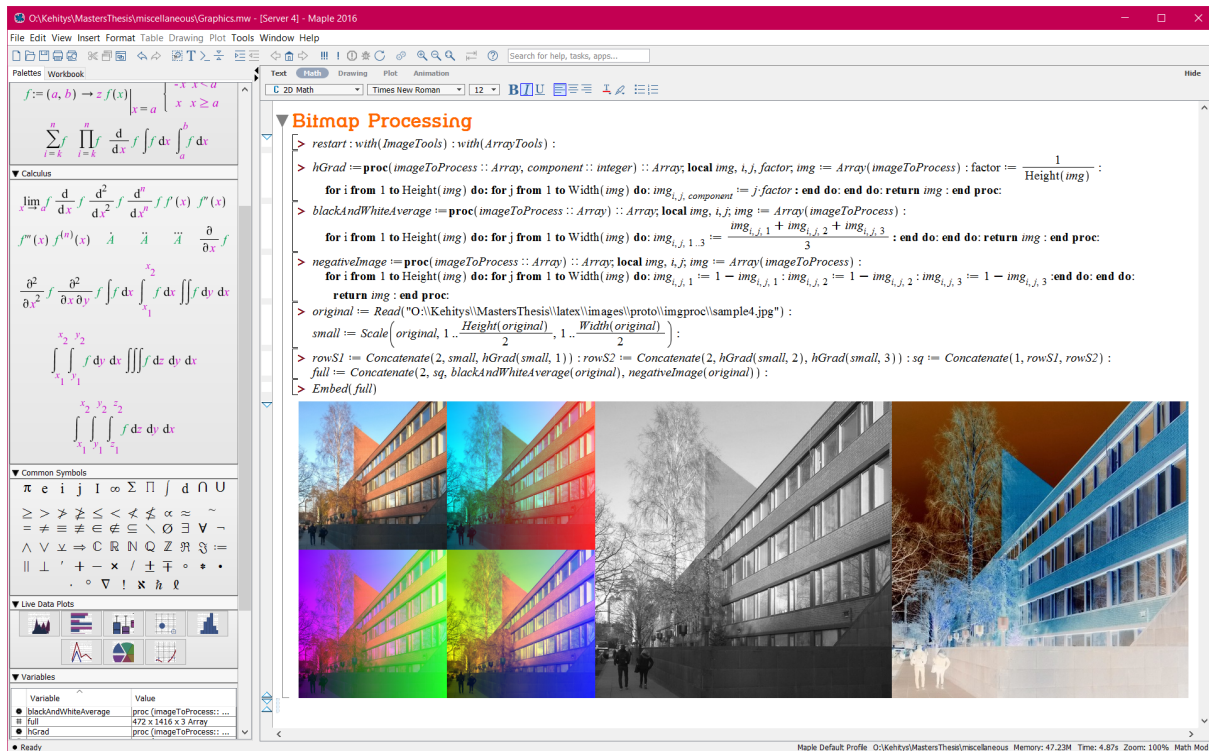


Figure 3.7. Trivial bitmap manipulation in Maple [WAT15A] version 2016.0 (Build 1113130). The program code in this figure is better presented in Code Example B.9.

programming, image processing and MATLAB itself is required to get the task done. As images are presented and treated as matrices, many elementary operations, as for instance different ways to combine images in relation to each other, must be implemented as matrix operations as there are no ready-made functions for them. Furthermore, some of the existing functions are not as refined as would be desired. For instance, there is a function to rotate an image, but at the moment its rotation algorithm produces only black background, as it fills the empty areas of the result image with zeros. One way to overcome this obstacle is to compute a mask image and use it to change the rotated image's pixels to some other colors. This, however, requires knowledge well beyond a student who is just learning how to call functions or how to use flow control structures, such as conditionals and loops.

Maple. MATLAB represents a transition between branches (2.2.2.2.3) and (2.2.2.2.1)—that is, from displaying media in separate windows to alternating code and media in a single window. Another example of the last-mentioned approach is Maplesoft Maple [WAT15A], which is a commercial mathematical software for symbolic calculations and in principle enables applying the Media Programming approach. The central part of its user interface (Figure 3.7) is a code editor that resembles the Live Editor of MATLAB: The content is based on cells that contain mainly either body text or executable input and that can be executed either one at a time or all of them in one batch. Maple has an extensive collection of palettes to help inputting mathematical content, which can be made to look somewhat like that of good mathematical textbooks.

As MATLAB, also Maple presents bitmaps as matrices and also it has a package for image processing. However, the facilities offered by that package are much more primitive than those of MATLAB's; for example, Maple's image rotation function supports only angles that are multiplies of 90 degrees. Thus Maple shares the MATLAB's problem

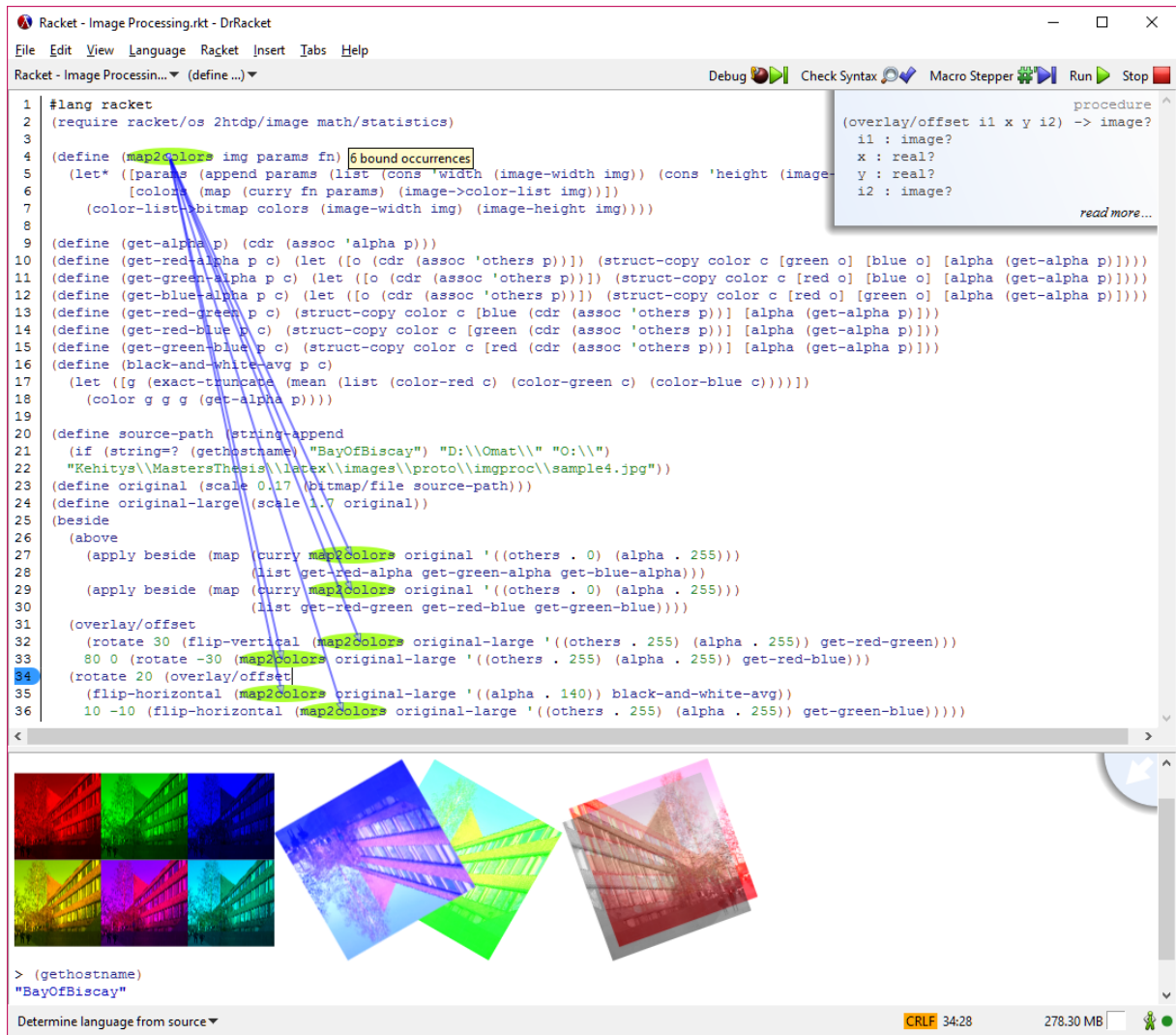


Figure 3.8. Trivial bitmap manipulation in DrRacket [RCK] version 6.4. The program code in this figure is better presented in Code Example B.10.

concerning introductory programming teaching: Any missing but necessary functionality must be implemented by the users themselves, but their knowledge level is not yet high enough to do it.

The name of the last branch of the contextualization approach classification to be discussed in this section—(2.2.2.2.2.2) *Everywhere*—refers to the name of its parent branch (2.2.2.2.2) *Media Is Part of Program Code*. In terms of media usage, the IDEs in this branch go even further compared to the ones in which program code and media alternate: The IDEs in this branch provide facilities to integrate media and program code together—not only in outputs but also inputs. In other words, programmer can write program code that is composed of both text and the types of media that the IDE in question supports. This section discusses two examples from this branch: DrRacket and Wolfram Mathematica.

DrRacket. The first example from the branch (2.2.2.2.2.2), DrRacket [RCK], is an open-source project developed by people from several North-American universities. It consists of a beginner-friendly IDE (see Figure 3.8) and a several tools to support development using a family of LISP-based programming languages, of which the primary one is called Racket. The Racket language was formerly called PLT Scheme, and the DrRacket IDE

was DrScheme, but with the release of version 5.0, they were renamed to avoid the burden of being associated with the Scheme language. There exist several different variations of the Racket language, some of which are aimed for people learning programming. There is also, for example, a variation that offers the Racket language with static typing. Racket also enjoys a number of code libraries, some of which are intended for beginners. What is more, some introductory programming books have been written based on Racket, the functionality presented in two of which [BLO13; FEL+15] is partly used as requirements for the prototype developed for this thesis (§ 4.2).

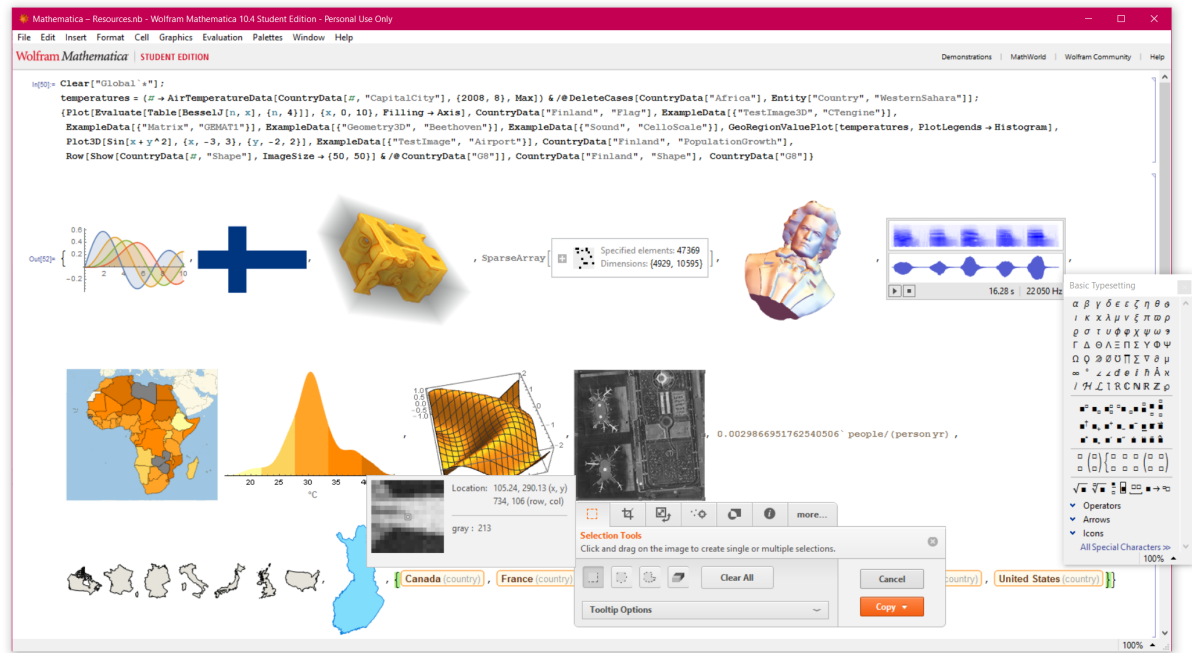
As can be seen from Figure 3.8, the IDE has two main portions: A code editor window—a *definitions area*—at the top and a REPL window at the bottom. There also exists a context-sensitive help displaying function signatures, and the upper editor is able to show some arrow-based visualizations, like relations of names (as in Figure 3.8 from a function definition to its uses) as well as the call stack (that is, activation records) from the top, where an error occurred, to the bottom, where the code the programmer wrote was started to be executed. Both the definition window and the REPL support embedded media, such as images and chunks containing XML and web pages (that is, HTML). As for instance in the Jython Environment for Students, the REPL also displays textual output and error messages from any code executed. In the figure, the program in the definitions area is run and the result of executing it is displayed in the REPL window, after which the statement (`gethostname`) is run in it.

DrRacket enables practicing at least image-based Multimedia Programming. The official distribution alone has libraries¹² for instance for turtle graphics, compositing primitive shapes and bitmaps, computing pixel-based bitmap effects, plotting graphs of mathematical functions, and three-dimensional graphics. However, even if using bitmaps is possible, it seems to be a side issue for the main image libraries, which focus extensively on internally vector-based graphics. Support for instance color model calculations and advanced bitmap effects seems currently to be missing or in libraries that are mutually incompatible. However, as Figure 3.8 shows, elementary bitmap processing functions, such as rotation and scaling, are implemented. Support for combining images is versatile.

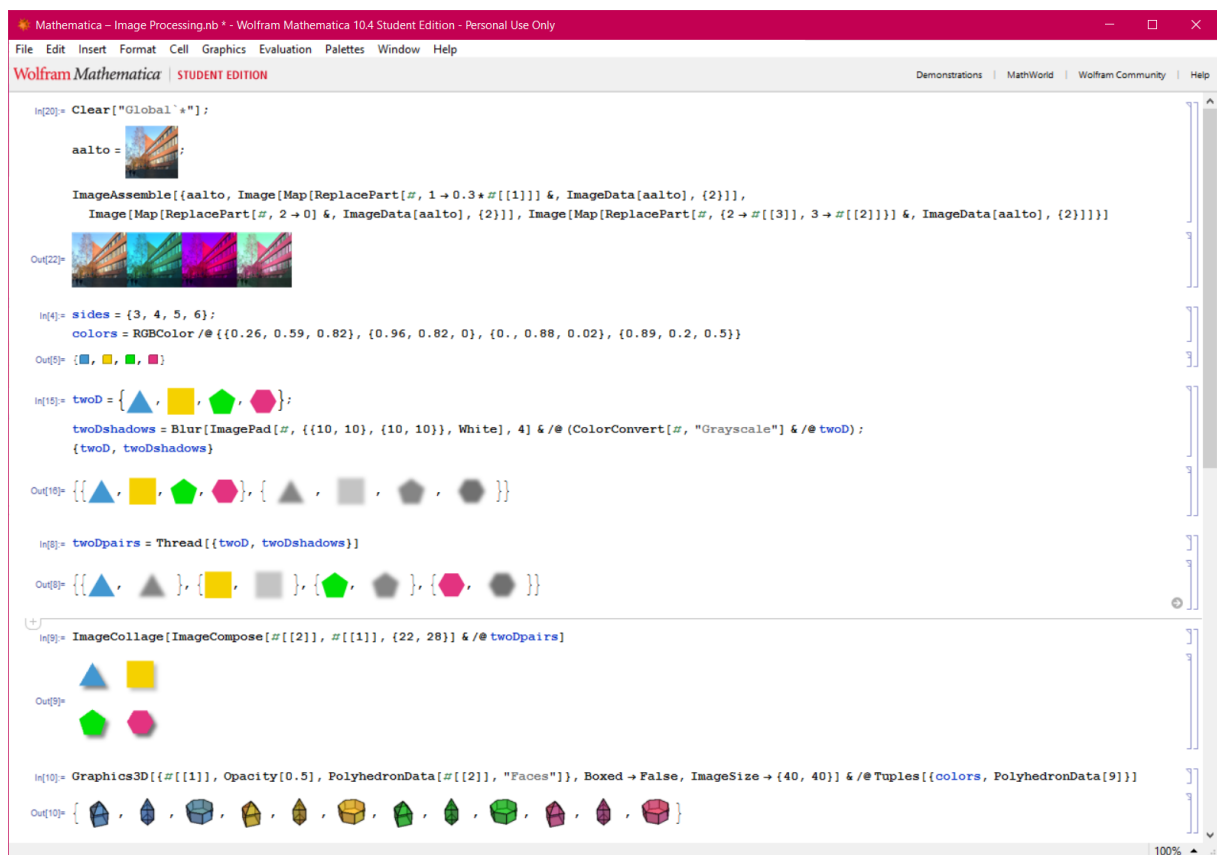
Mathematica. The another example from the branch (2.2.2.2.2), Wolfram Mathematica [WOL15], is a commercial application for performing symbolic mathematics, and seems to be the current state-of-the-art what comes to combining program code and media. In Figure 3.9a, the program code at the top of the window results in a list that—in addition to text—contains for example a sublist, transformable 2D and 3D vector models, applet-like GUIs presenting a numerical matrix and a sound waveform, and a bitmap image that at the moment is being edited. Whereas DrRacket’s implementation of displaying images in the code editor is functionally quite a simple one, in Mathematica images are first-class applet-like objects that can be resized by dragging and edited via pop-up menus and windows. In the same fashion, for instance 3D-models can be freely rotated by dragging as well as sound waveforms played by clicking their play-buttons. While these objects are resized, the code editor updates its contents on the basis of its own size and the sizes of its content elements. Images can also be dragged into the editor.

The cell-based code editors of Mathematica, Maple, and MATLAB’s Live Editor are very alike: Again, cells can contain text or input for Mathematica, and execution results for any input cells are displayed below of those cells. The extent of Mathematica’s bitmap processing support competes with that of MATLAB’s Image Processing Toolbox, and on some aspects it excels clearly over it. To backtrack to the bitmap rotation exam-

¹²For instance: `2htdp/image`, `2htdp/planetcute`, `graphics/turtles`, `images`, `pict`, `plot`, `pict/-snip`, `picturing-programs`, and `racket/draw`.



(a) Multimedia integrated into the code editor as first-class citizens. The program code in this figure is better presented in Code Example B.11.



(b) Trivial bitmap and list manipulation. The program code in this figure is better presented in Code Example B.12.

Figure 3.9. Wolfram Mathematica [WOL15] version 10.4.0.0.

ple given when discussing MATLAB, Mathematica produces rotated bitmaps with black background by default. However, changing that is a simple thing of specifying a parameter like `Background→White`. There is also a good selection of beginner-friendly functions for simple manipulations, such as basic transformations, artistic effects, and compositing bitmaps into collages. Pixel-based bitmap operations are also supported in several different ways, such as in procedural way by looping over pixels as well as in functional way by giving a self-programmed function to Mathematicas `ImageApply[]` function.

The IDEs discussed in this section hopefully both clarify the interpretation of the contextualization approach classification in § 2.3 and provide a quick review of the kinds of IDEs that support Media Programming. From the viewpoint of this thesis, however, a much more important goal is to get views on the best way to implement the prototype that will be covered in the next two chapters. In this regard, a code editor that works with Scala but offers the user interface of Mathematica’s code editor would be the ultimate goal to pursue.

Libraries. The other major branch of tools providing support for Media Programming in the classification of § 2.3 is (2.2.1) *Program Code Libraries*. They are further divided into two branches: The ones under the branch (2.2.1.1) *Media Input/Output Independent from IDE* take care of their media input and output needs “independently” without any help from the development environments. The second branch, in turn, (2.2.1.2) *Media Input/Output Together with IDE*, contains libraries that can take advantage of input and output facilities offered by some IDEs.

Examples of libraries of this kind are for instance the so-called Guzdial–Ericson Multimedia Class Library (the `java-sources.zip` at [MCO]) and the Processing library [PRC; CAR10], which both are open-sourced and offer Java-based APIs. The former is tailored for Guzdial’s and Ericson’s Media Computation curriculum, in which capacity it offers scaffolding for creating and simple editing for instance turtle graphics, bitmaps, sound waveforms, as well as videos. The latter one, in turn, is targeted to interactive drawing-based graphics, such as generative art, made by for instance graphic designers, hobbyists, and other unprofessional programmers.

In addition to the products that can be perceived as traditional code libraries, there are ways to use other applications effectively as libraries. One such a way is so-called Mediascripting [RDW13]—an interesting variation of Guzdial’s and Ericson’s Media Computation curriculum. In it students do not program stand-alone programs but plug-ins to an existing graphical application—in this case, the GIMP (GNU Image Manipulation Program) [GMP], which provides APIs for several programming languages, such as Python and Scheme. In this way, students have not only a complete framework inside of which to develop and test their programs, but also the full power of GIMP’s own image manipulation API at their disposal, which in turn enables achieving more advanced creative and artistic results than with stand-alone programs supported by a simple bespoke graphics library. This approach could also appear as a more useful one, because when students provide self-developed custom functionality into a commonly-used application, they see that even with relatively low programming skills they still can affect for instance by automating tedious tasks and by creating completely new kinds of effects. As discussed before, both of these points can increase students motivation to study more and harder.

This concludes the discussion of both Media Programming and the related theoretical matters in the scope chosen for this thesis. The next two chapters deal with the research prototype developed for this thesis—the fourth one with its functionality and the fifth one with its implementation.

Chapter 4

Prototype: Media Computation in Scala

The third chapter explored the Media Programming approach and the one before that motivation and contextualization. Now is the time for the discussion to transition towards the research prototype that was referred to in the [RG1](#).

This chapter starts with discussion of the prototype’s stakeholders (§ 4.1) and requirements (§ 4.2). The next topic, a brief overview of the integrated development environments currently available for Scala (§ 4.3), lays ground for the discussion concerning both the prototype’s high-level structure (§ 4.4) and functionality (§ 4.5). Finally, after presenting the prototype, the chapter demonstrates its usage with some exercise examples (§ 4.6). The next chapter is concerned with the implementation side of the prototype.

4.1 Stakeholders

Regarding the research prototype’s implementation, there are three important stakeholders to consider; They are **Instructor**, **Student**, and **Developer/Maintainer** (Figure 4.1). To start with, **Instructors** are constantly seeking and researching ways to make their teaching more efficient and enjoyable, and are interested to try the media computation approach via integration of this prototype into their course materials, demonstrations and exercises. Furthermore, they will also assess its applicability and efficiency from the pedagogical viewpoint.

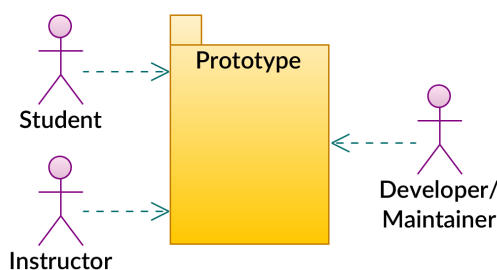


Figure 4.1. The most important stakeholders of the prototype.

Students are enrolled to courses held by **Instructors** as well as doing exercises, hopefully primarily because they are interested in the topics of the course in question, but also due to the fact that some courses are mandatory—interesting or not. Excluding **Students** who, for instance, are just familiarizing themselves with the course to decide if they will take

it or not, **Students** generally want at least to pass the course regardless if they learn or not. However, one should hope they wish to learn well the topics they are being taught. To help their learning, they are encouraged or required to use this prototype to complete exercises and practice freely.

Finally, **Developer/Maintainer** is responsible of implementing the prototype and maintaining it. This involves considering everything related to requirements, development tools, programming, testing, documentation, delivery, and maintenance with the goal of producing the most optimal solution with the resources available at the time.

In addition to the previous three primary stakeholders, other ones can be identified as well. For example, the schools and universities as institutions are indirectly interested in the results of their teaching as well as the tools used to achieve them. The prototype may also be of interest for non-students who wish to learn programming or update their skills to include the Scala language. Finally, third-party developers may wish to use or support the prototype in their products.

4.2 Requirements

While planning the prototype to be implemented, to keep the amount of work in check, it was decided to narrow its scope down by including only features related to creating and processing static bitmaps. Sound, videos, charting, open data, QR-codes, and so on were intentionally dropped from the scope. The full set of functional requirements for the prototype is listed in the Table 4.1. There are only a few of them, but as many of them in their full effect are very tedious or practically impossible to implement in the scope of this Thesis, there still was practically no end for the functionality to be implemented.

The Scala IDE's REPL client was selected for the front end [R1](#) because it is the chosen development environment of the Aalto's Scala courses discussed in § 1.2. The possibility of exploring ways to represent chunks of media (here, static bitmaps) directly inside the REPL as first-class citizens [R2](#) has been one of the driving forces behind the prototype. Further desirable characteristics for that feature would be (1) the ability to copy and paste media using the system clipboard, (2) the possibility to load media from files and save it into them via graphical user interface, and (3) presentation of media to users as embedded controls providing interactive functionality to fiddle with.

Considering the classification presented in Table 3.1, the goal of the prototype was to support Media Programming according to the approaches of both (2.1) *Filtering Information* and (2.2) *Compositing Information*, but as mentioned in § 1.4, only for bitmaps. Thus, the more accurate categories in the classification are (2.1.1) *Pixel-Based Bitmap Filtering* [R8](#) as well as both (2.2.1) *Primitive Shapes* → *More Complex Shapes* and (2.2.2) *Bitmaps* → *Visual Collages* [R6](#) [R7](#).

What comes to the desired quality attributes of the prototype, [R9](#) was the only one to be explicitly defined and recorded. Of course, general objectives like ease of installation and use, rapidity of operation, and low memory consumption were considered, but as none of them were defined or measured in any meaningful and generalizable way in the scope of this thesis, they will not be discussed here any further.

Realization of these requirements is discussed in § 6.1 later, and the pilot study to assess the fulfillment of [R9](#) (see § 1.3) is described in § 6.2.

4.3 Professional IDEs for Scala

To lay foundations for discussion of the prototype’s functionality and implementation, this section very briefly introduces three IDEs, JetBrains IntelliJ IDEA, Eclipse, and NetBeans, targeted for professional Scala-based development.

IntelliJ IDEA [JET15] is a commercial professional IDE, although it is free for students. It offers plug-in development APIs that can be used for example to provide support for new languages—this is how also its Scala support has been implemented. IntelliJ IDEA’s Scala plugin offers the usual code editing and analyzing facilities, as well as a REPL and a worksheet-style code editor. The basic components of IntelliJ IDEA should be quite familiar (Figure 4.2). The upper-left window displays the project content, at the center there are code editors, and the upper-right window presents build management functionality. The window with blue background is the code completion window opened from the top-most code editor, and below it there is a REPL window for Scala. The bottom-most window behind the REPL displays any to-do items recognized from source code’s comments. The buttons on the sides of the main window open more facilities related to for instance project structure, version control, and problems found in the program code.

Table 4.1. Functional requirements for the prototype implemented. As the primary customer, Instructor stakeholder owns all of the requirements listed. Since these requirements were gotten practically ready-made and reflect the needs of teaching, exploration of the deep reasons behind them is omitted in the scope of this Thesis.

ID	Description
R1	The prototype’s primary user interface should be Scala IDE’s REPL client.
R2	The prototype should at least embed bitmaps into Scala IDE’s REPL client, as is done in the user interface of DrRacket’s IDE [RCK], preferably enabling users to handle them like characters, or images in a word processing application.
R3	The prototype should be designed and packaged in a way that makes the media-related functionality usable as a class library in other REPL implementations/clients than Scala IDE’s, such as a REPL being run in a console window unable to display bitmaps.
R4	The primary programmable objects the prototype provides to its users should be immutable.
R5	The prototype should support both object-oriented (<code>object.method(parameters)</code>) and function-based (<code>function(object, parameters)</code>) programming.
R6	It should be possible to perform bitmap-related examples presented in the Chapter 1 of <i>Picturing Programs</i> [BLO13, pp. 9–24].
R7	It should be possible to perform bitmap-related examples presented in the Prologue of <i>How to Design Programs</i> [FEL+15].
R8	It should be possible to perform most of the bitmap-related examples presented in <i>Introduction to Computing & Programming with Java</i> [GE05, pp. 38–211].
R9	By using the prototype, teachers should be able to design demonstrations and exercises they find pedagogically useful.

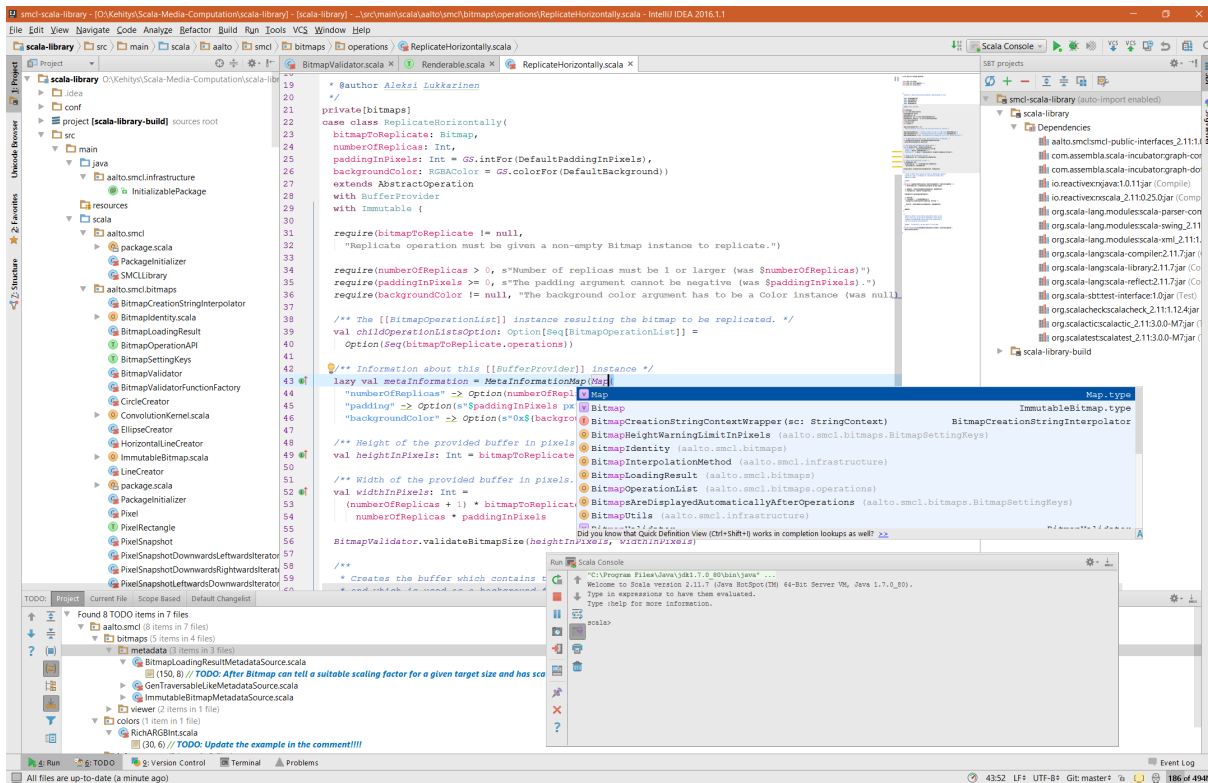


Figure 4.2. JetBrains IntelliJ IDEA Ultimate 2016.1.1 (build IU-145.597) application window with JetBrains’ Scala plugin version 3.0.2 on Windows 10.

Eclipse [ECL15], in turn, is an open-source IDE project originated in the beginning of the 2000s. Previously it has been targeted mainly for Java-based development, but after its launch, its infrastructure has evolved to be a development platform, for which the Java Development Tools (JDT) that provides support for Java-based development is only one set of plug-ins. As with IntelliJ IDEA, support for Scala is implemented as a new set of plug-ins called Scala IDE [SID15], which is an open-source project separate from Eclipse itself. Furthermore, to complement the Scala IDE’s functionality, a second plug-in component called Scala Worksheet [ScW] has been developed. The primary parts of Eclipse’s user interface are visible in Figure 4.4. At the top-left corner of the application window the Package Explorer displays contents of the projects in the current workspace, and the customized REPL window is on the right. Between them are the open files of the project, and at the bottom-left corner there is an text file open to act as a memo pad. Eclipse and its plug-ins offer numerous other windows for various purposes, but they are hidden in the figure.

NetBeans [NTB] is a third popular professional-level IDE, the development of which started in 1996 in Czech Republic as a student project that led to a start-up company developing NetBeans [NBH]. That company was bought by Sun Microsystems, which open-sourced NetBeans in 2000 (before being acquired in 2010 by Oracle Corporation). When considering their basic functionality, both Eclipse, NetBeans, and JetBrains IntelliJ IDEA are quite similar and compete for users with each other. Because of these similarities, and as the basic facilities of professional IDEs are expected to be very familiar for the reader, NetBeans is not discussed here further. The important thing in the context of this thesis is that the REPL windows’ technical implementations of all three IDEs support only text. Finding possibilities to improve this situation is one of the purposes of the research prototype built for this thesis, and is also expressed in R2.

4.4 Structure

The research prototype is composed of three independent components (Figure 4.3): *Scala Media Computation Library* (SMCL), *Customized Scala IDE REPL*, and *SMCL External Interfaces*. As can be seen from the figure, the front end of the prototype is composed of customized Scala IDE’s REPL and the external interfaces, whereas SMCL itself together with the external interfaces forms the back end.

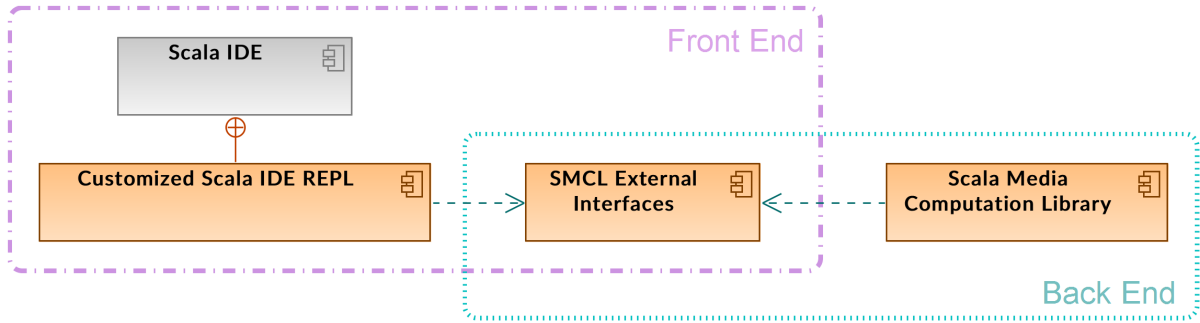


Figure 4.3. The main components of the prototype, grouped into front end and back end.

The *Customized Scala IDE REPL* is the simplistic REPL client that is bundled with Scala IDE, but with a few modifications, as described below in § 4.5.1. This means, that for the prototype to work as intended, a tailored version of the full Scala IDE project has to be built. Both the *Scala Media Computation Library* and the *SMCL External Interfaces* are independent Scala-based class libraries that are developed for this thesis from scratch. The former offers functionality to support Media Programming and the latter one provides a link between SMCL and a REPL client, which obviously has to support the *SMCL External Interfaces* library. That link makes it possible for the front end to query metadata to display in the context of any objects the Scala interpreter returns as responses to executed commands.

4.5 Functionality

The research prototype has three touchpoints between its users and itself: The API of the SMCL; the bitmap viewer that SMCL can use if a REPL that supports bitmaps and the *SMCL External Interfaces* are not available; and the *Customized Scala IDE REPL*, if it is being used. This section describes the functionality of these touchpoints.

4.5.1 Scala IDE’s REPL

As one of the primary requirements for the prototype, [R2](#), was that it should display the processed bitmaps as images inside of itself, the history list of the Scala IDE’s REPL client was modified to enable alternating code and bitmaps (Figure 4.4) in the fashion of Maple and MATLAB’s Live Editor. However, it is not currently possible (1) to give bitmaps as input via the command line, (2) to display code and bitmaps integrated together, nor (3) to copy or paste bitmaps either to the REPL or from it.

In addition to the capability of displaying bitmaps, two other modifications to the Scala IDE’s REPL were made. The first of them is that a single REPL window is not any more associated with only one of the projects in the presently open workspace. Instead,

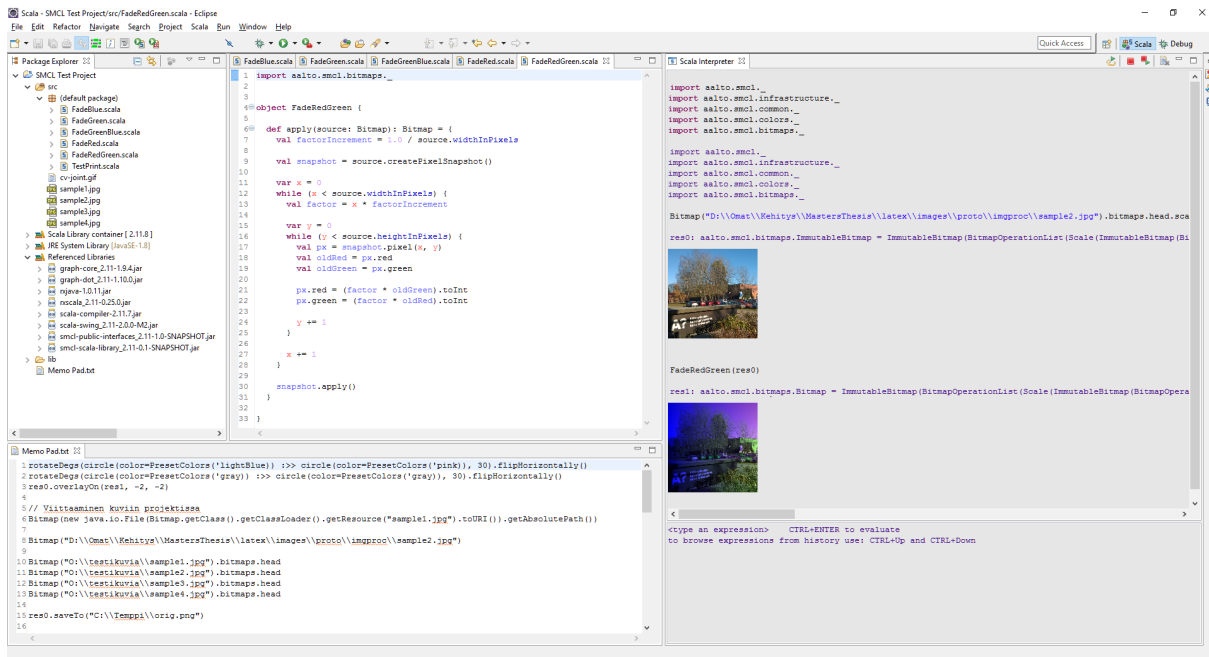


Figure 4.4. The Eclipse Luna SR2 (version 4.4.2) window with customized pre-release version of Scala IDE 4.5.0 on Windows 10 after starting the customized Scala IDE REPL (the *Scala Interpreter* tab).

the phase for choosing a project was removed from the UI and the REPL’s classpath now includes all of the projects available at the moment the REPL is launched. The obvious effect of this is that all content of the default packages are directly accessible and any package in any project can be used after it is imported. The theoretical trade-off of this modification is a possibility for overlapping names, but when this behavior is acknowledged, the project templates Students will use for exercises can be prepared accordingly. As a positive side, this modification increases both flexibility and simplicity, and also presents the teacher a possibility to emphasize the need for visibility scopes, such as namespaces and packages.

The third modification to the Scala IDE’s REPL was that when it is launched, it automatically imports the SMCL packages, the content of which is necessary for the most use cases (see Figure 4.4 above). This modification saves Students the trouble of remembering and writing the necessary imports every time they launch a new REPL.

4.5.2 SMCL

The fact that this prototype is intended to be used by Students who have no previous programming experience means that it should be as effortless to use as possible (§ 3.4). In SMCL’s API this is answered to by trying to construct it in such a way that

- no separate library initialization commands are necessary
- no explicit object creation with the `new` keyword is necessary, and
- no secondary auxiliary objects, such as settings, state, or static function containers should be needed to be used and passed around.

All in all, the command line user experience of SMCL should correspond with that of traditional text-based command-line interfaces (CLI) of shells and command prompts provided by operating systems: Once a shell is launched, it is immediately possible to process files and other resources with the available commands without using imports, initializations,

```

1  GS.list()

3  Integers:
4  - DefaultArcStartAngleInDegrees: 0
5  - BitmapHeightWarningLimitInPixels: 800
6  - ColorVisualizationTileSideLengthInPixels: 80
7  - DefaultRoundingHeightInPixels: 20
8  - DefaultBitmapWidthInPixels: 50
9  - DefaultPaddingInPixels: 5
10 - DefaultCircleRadiusInPixels: 10
11 - DefaultRoundingWidthInPixels: 20
12 - DefaultBitmapHeightInPixels: 50
13 - DefaultArcAngleInDegrees: 180
14 - BitmapWidthWarningLimitInPixels: 800

16 Booleans:
17 - NewBitmapsAreDisplayedAutomatically: false
18 - ShapesHaveFillingsByDefault: false
19 - BitmapsAreDisplayedAutomaticallyAfterOperations: false
20 - CanvasesAreResizedBasedOnTransformations: true
21 - ShapesHaveBordersByDefault: true

23 Enumerations:
24 - DefaultVerticalAlignment: Middle
25 - PlatformBitmapInterpolationMethod: NearestNeighbor
26 - DefaultHorizontalAlignment: Left

28 Colors:
29 - DefaultBackground: ARGB: 0x00ffffff -- 0 - 255 - 255 - 255
30 - DefaultSecondary: black (ARGB: 0xff000000 -- 255 - 0 - 0 - 0)
31 - DefaultPrimary: black (ARGB: 0xff000000 -- 255 - 0 - 0 - 0)

```

Code Example 4.1. The global settings and their default values.

settings, and artificial container classes for commands (methods) and values (attributes). For instance, `Student` should be able to write `pwd()` instead of `import settings; import unixfs; val gs = GlobalSettings(); val fs = UnixFileSystem(); UnixFileUtils.pwd(fs, gs)`. Only the primary objects, such as bitmaps and colors, are acceptable to be passed around, as they are actually processed with the prototype.

Settings. To avoid the use of setting objects, a global setting registry was implemented. The idea was to extend Eclipse’s setting dialog to enable altering the values contained by it, but because of the limited time frame of this work, that extension had to be abandoned. The setting registry is accessible through an object called `GS` (*global settings*). It was designed from the viewpoint of `Developer/Maintainer` and is not very user-friendly for `Student`, but the values contained by it were not thought to be modified by `Students`. Instead, `Instructors` would provide a configuration that would fit to their exercises. If `Students` are to modify the settings in the future by using program code, some kind of wrappers could be generated for them to make them more easily accessible.

Currently there are four types of settings—integers, booleans, enumerations, and colors. The actual settings and their default values can be listed with `GS.list()`, as is done in Code Example 4.1. The value of an individual setting can be queried for instance with `GS(NewBitmapsAreDisplayedAutomatically).value`, and the values can be set in the fashion of `GS.booleanSettingFor(NewBitmapsAreDisplayedAutomatically).value = true`. A more detailed description of the settings system is given in § 5.1.3 with the class model presented in Figure 5.3.

Colors. The prototype presents colors in the RGB (red, green, blue) color space, augmented with opacity/transparency information. For calculations, they can be converted between the HSI (hue, saturation, intensity), HSL (hue, saturation, lightness), and RGB color spaces, but the prototype currently offers no classes to represent colors in HSI



Figure 4.5. The prototype displays all colors the way illustrated by this figure: One rectangle full of the color in question itself as well as three other rectangles for displaying the color on black, gray and white background. These four metadata images are created by SMCL on request and passed through *SMCL External Interfaces* to the REPL console for displaying them to the user. As can be seen, the color used in the figure was retrieved from the preset color map.

or HSL (or other) color spaces. The `RGBAColor` and the `RichRGBAColor` together offer methods for deriving new colors for instance by

- setting and adjusting individual color components (red, green, blue, alpha) by percentages/factors
- throwing away—i.e., zeroing—specific color components and keeping the rest
- shading and tinting colors by percentages/factors
- circling the RGB color wheel by a given angle, and
- converting RGB colors to grayscale using given weights for the color components.

In addition to the user-defined colors, the `PresetColors` map contains pre-defined color objects for almost all of the color names defined in the section 4.3 of the CSS Color Module Level 3 [WOR11] standard; these colors can be retrieved as is shown in Figure 4.5. The figure also shows how all `RGBAColors` are displayed in the customized Scala IDE REPL. A more detailed description of the color-related classes is given in § 5.1.4 with the class model presented in Figure 5.4.

Bitmaps. The class used to represent all bitmaps in the prototype is called `ImmutableBitmap`. There is also an “alias” for it called simply `Bitmap`, because the default kind of all bitmaps is thought to be the immutable one (of course, mutable bitmaps are not currently implemented at all). Bitmaps can be created in various ways. One of the ways is to create bitmaps for primitive shapes using functions such as `circle()`, `ellipse()`, `hLine()`, `line()`, `rectangle()`, and `rRectangle()` (with rounded corners). Another one is to load bitmap files using the `Bitmap` object, which can also be done using a string interpolator in `bmpf"/path/to/image.png"` fashion. Also, every operation performed for any bitmap creates a new bitmap—they are immutable, after all.

The `ImmutableBitmap` and its associates offer many ways to composite and filter bitmaps. For example, they can be

- composited next to each other with operators (e.g., `:>>`, `:<<`, `:/\`, `:\`) and functions (e.g., `appendOnLeft()`, `appendOnTop()`, `replicateHorizontally()`)
- composited by overlaying them with operators (e.g., `:|+|`, `:|*-|`) and functions (e.g., `overlayOn()`, `underlayBehind()`, `overlayPerAlignments()`)
- cropped, trimmed, scaled (e.g., `scale()`, `scaleHorizontally()`), rotated (e.g., `rotate90DegsCw()`, `rotateDegs()`), sheared (`shear()`, `shearVertically()`), and flipped (e.g., `flipHorizontally()`, `flipDiagonally()`), and
- cleared, negated, posterized, as well as converted to grayscale by lightness, luminosity, or by specified weights for individual color channels.

In addition, their individual color channels—or combinations of them—can be zeroed and negated, and their canvases can be enlarged (`augmentCanvas()`). Moreover, primitive

shapes can be drawn on them, and their individual pixels can be processed using pixel snapshots and the related pixel iterators (see § 4.6.2). Due to the limited time frame of this work, SMCL does not provide more advanced bitmap effects, but bitmaps can also be convoluted using self-specified kernels for instance for blurring, sharpening, embossing, and simple edge detection. A more detailed description of the bitmap-related classes is given in § 5.1.5 with the class model presented in Figure 5.5.

Metadata. About any data processed with SMCL, the REPL client’s history window should be able to display metadata, such as text, bitmaps, videos, and editable objects resembling applets. To make it possible, the *SMCL External Interfaces* (Figure 4.4) provides a way for the REPL client to query SMCL for the metadata that should be displayed for a specific object. At the moment, the REPL client can display only bitmaps, and SMCL is able to produce bitmap representations of both bitmaps themselves (obviously) and colors (Figure 4.5). This functionality could be extended to visualize more objects and to provide more detailed and structured metadata, including the integration of media and program code in outputs. However, the REPL client of this prototype is capable of displaying only simple bitmaps below the textual execution results.

Bitmap Viewer. SMCL contains a small GUI “application” (Figure 4.6) for displaying bitmaps. In SMCL, every bitmap contains an identifier that is unique in the set of “new bitmaps” created in a single REPL session. When “a new version” of an existing bitmap is created, the identifier is inherited by that “new version”. Every time a new viewer window is opened, it is associated with the identifier of the bitmap for which it was opened. Whenever a bitmap asks the viewer application to be displayed, the application checks if a viewer window for the identifier of the asking bitmap is already created. If one exists, it is updated, and if it was closed, it is opened again. If there is no viewer for the bitmap identifier in question, a new window is opened to display the bitmap.

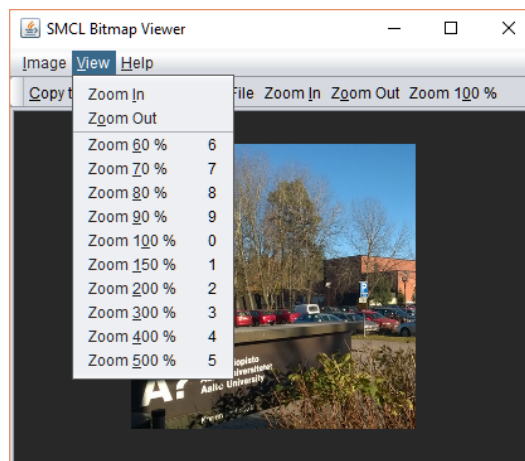


Figure 4.6. SMCL’s bitmap viewer on Windows 10.

To control the functionality described above, the bitmaps in SMCL respect two options, according to which they can be set to ask to be displayed automatically both when they are created without an inherited identity (a “new bitmap”) and when an existing identity is inherited (“a new version”). If they are not displayed automatically, a single bitmap can be displayed using the bitmap’s `display()` method. It is also possible to close all viewer windows with a single command instead of clicking them through with the mouse.

As can be seen from Figure 4.6, the bitmap viewer window is a very simple one. Currently, bitmaps can only be zoomed by fixed percentages. The intention was to enable

both copying them to the system clipboard as well as saving them into files, but these features were left unimplemented because of the limited time frame of the work. The zoom feature also zooms always towards the upper-left corner of the image—not towards the area that is being viewed or on which the mouse cursor currently is.

Now, as the relevant functionality of the prototype is presented, it is time to move on. The next section presents some examples that make use of the research prototype’s features for both compositing and iterating over pixels.

4.6 Exercise Examples

This section presents some short examples of using the research prototype in creating simple exercises applicable for an introductory programming course. While their main purpose is to teach programming, they also require and hopefully strengthen skills and knowledge, such as bitmap processing, mathematics, creativity, and hopefully, transferring and applying any acquired knowledge and skill broadly to many kinds of problems and situations [for instance: ILL09, p. 17; DAN+12]. While the learning goals of a course dictates the content to be contextualized, more of both creative inspiration and theoretical knowledge of graphic as a context can be found for instance from books¹ about topics such as general graphical design, chromatics, information graphics, signs, logos, typography, layout, packages, business cards, patterns, photography, and generative art. These examples are mathematically quite trivial—the required knowledge level lies somewhere in between the 7th and the 12th grade. Thus, assuming that these exercises are targeted to university students, applying them obviously should not require additional study of mathematics at all.

4.6.1 Expressions and the Little Quilt Language

To illustrate expressions in his book *Programming Languages* [SET97], Sethi designed a simple language that was expressed via functions. The purpose of this language was to enable compositing fabrics out of small quilts—hence the name *Little Quilt*. Code Example 4.2 presents an example, which defines two primitive quilts of 50×50 pixels—the first of them contains arcs and the second one straight lines—and then uses them to build the two bigger ones that are presented in Figure 4.7. To have the quilts sewn together, the padding that is left between bitmaps by default is zeroed (line 13). In the classification presented in Table 3.1, this example would be placed under the branch (2.2) *Compositing Information*, either in (2.2.1) *Primitive Shapes* \rightarrow *More Complex Shapes* or (2.2.2) *Bitmaps* \rightarrow *Visual Collages*.

For this example, the operations `sew()` and `turn()` were added into SMCL as aliases for the existing operations `appendHorizontally()` and `rotate90DegsCw()`, respectively. Actually, the functions `unturn()` and `pile()` that are defined in the example on the basis of the primitive operations `turn()` and `sew()`, exist already as aliases for `rotate90DegsCcw()` and `appendVertically()`, respectively. In the code listing they are left without highlight to emphasize their redefinition in the example.

¹For instance: [BGL12; COL12; FAW12; EBK12; REN12; SM12; YAC12; TON09; PRÄ06; SAM07; MAL07; SAM04; MP06; SAM06].

```

1  val c = PresetColors('seaGreen')
2  val a = Bitmap(initialBackgroundColor=PresetColors('wheat'))
3      .drawCircle(-10, 60, 64, color=c)
4      .drawCircle(3, 47, 52, color=c)
5      .drawCircle(3, 47, 58, color=c)

7  val c = PresetColors('sienna')
8  val b = Bitmap(initialBackgroundColor=PresetColors('lightYellow'))
9      .drawLine(12, 0, 50, 38, c)
10     .drawLine(25, 0, 50, 25, c)
11     .drawLine(38, 0, 50, 12, c)

13  GS.intSettingFor(DefaultPaddingInPixels).value = 0

15  def unturn(x: Bitmap): Bitmap = turn(turn(turn(x)))
16  def pile(x: Bitmap, y: Bitmap): Bitmap = unturn(sew(turn(y), turn(x)))

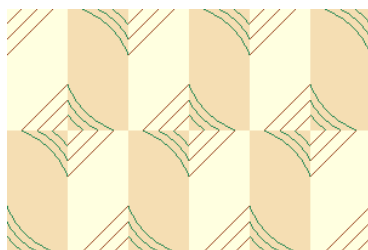
18  val aa = pile(a, turn(turn(a)))
19  val bb = pile(unturn(b), turn(b))
20  val p = sew(bb, aa)
21  val q = sew(aa, bb)
22  val r = pile(p, q)
23  val quilt1 = sew(sew(r, r), r)

25  val bb = pile(turn(b), unturn(b))
26  val ba = pile(unturn(b), turn(a))
27  val c_nw = sew(bb, ba)
28  val c_ne = turn(c_nw)
29  val c_se = turn(turn(c_nw))
30  val c_sw = unturn(c_nw)
31  val p = pile(turn(a), unturn(a))
32  val q = pile(turn(turn(a)), a)
33  val top = sew(sew(c_nw, p), sew(q, c_ne))
34  val bot = sew(sew(c_sw, q), sew(p, c_se))
35  val quilt2 = pile(top, bot)

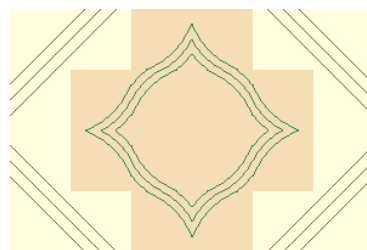
```

Code Example 4.2. Creation of the two quilts Sethi used as examples in his book [SET97, pp. 306, 378–379]. The final results of this example are presented in Figure 4.7.

Naturally, for more flashy results than those of Figure 4.7, one could load some bitmap to be used as a background texture for the two primitive quilts, or one could use bitmaps created entirely in some graphics software.



(a) quilt1.



(b) quilt2.

Figure 4.7. The final results of Code Example 4.2: Two quilts composed of $6 \times 4 = 24$ smaller quilts (300×200 pixels in total).

Considering [R5], this example made use of the function-based style of programming. The examples in the next section represent the object-oriented approach as well as a different genre of bitmap processing.

4.6.2 Repetition Structures and Pixel Iteration

The purpose of the examples presented in this sub-section is to practice creating repetition structures—both *for loops* and *for comprehensions* as well as *while loops*. Also, one of these examples explicitly uses *Iterator* design pattern [e.g., GAM+95, pp. 257–271] as a client—the iterator classes themselves are part of SMCL. The contextual motivation here is to creatively filter existing bitmaps by iterating through them pixel-by-pixel and adjusting their color components on the go. In the classification presented in Table 3.1, this would make the examples of this section to be placed under the branch (2.1.1) *Pixel-Based Bitmap Filtering*, and more specifically, in (2.1.1.1) *Color Adjustments*.

The examples are unoptimized and are given only to illustrate the usage of SMCL in teaching several program language structures. All of them demonstrate different usage possibilities of the `PixelSnapshot` class, an instance of which can be requested from each `ImmutableBitmap`. The execution results of these examples are visualized in Figure 4.8.

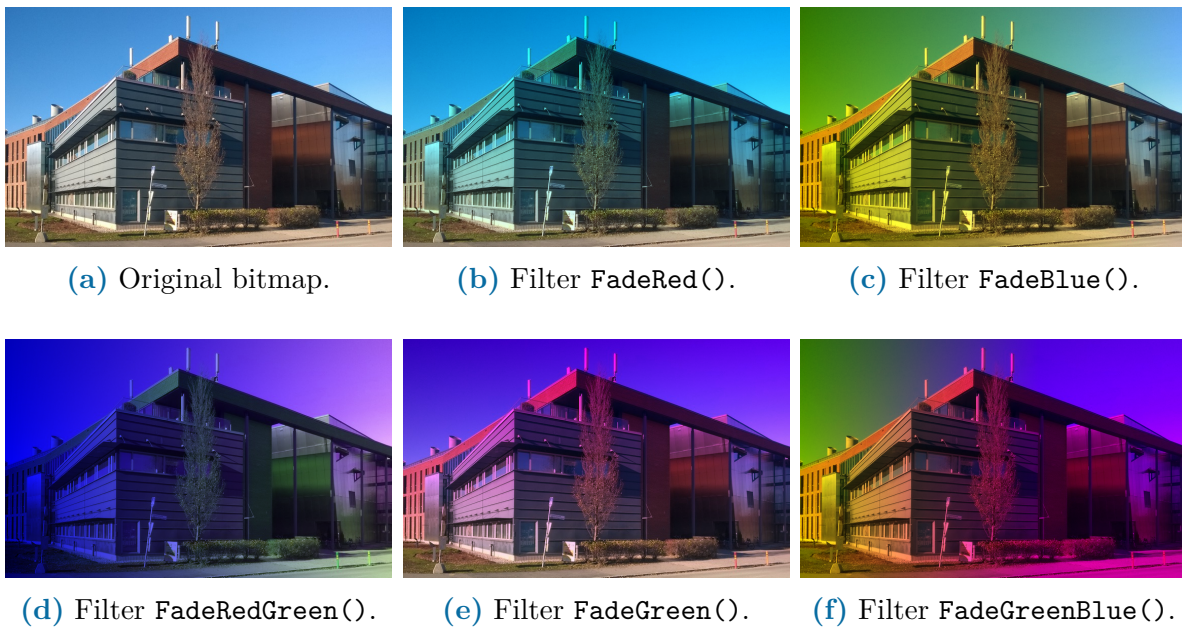


Figure 4.8. Results of applying the pixel iteration examples presented in this section.

Code Example 4.3 creates such a vertical color gradient that at the topmost line the amount of red is zero, and it increases line by line downwards as a percentage of the original red amount of each pixel, until at the bottom the amount of red (100 %) corresponds the original bitmap. The key programming-related concept in this example is *for loop*. The result is illustrated in Figure 4.8b.

```

1  import aalto.smcl.bitmaps._

4  object FadeRed {

6      def apply(source: Bitmap): Bitmap = {
7          val factorIncrement = 1.0 / source.heightInPixels

9          val snapshot = source.createPixelSnapshot()

11         for (y <- 0 to source.heightInPixels - 1; x <- 0 to source.widthInPixels - 1) {
12             val px = snapshot.pixel(x, y)
13             px.red = (y * factorIncrement * px.red).toInt
14         }

16         snapshot.apply()
17     }

19 }

```

Code Example 4.3. Pixel iteration using `PixelSnapshot` and *for loop*. The execution result of this example is illustrated in the Figure 4.8b.

Code Example 4.4 adjusts the amount of blue in each pixel so, that on the leftmost column of pixels, there is no blue at all, while the amount of blue increases rightwards as a percentage of the original amount of each pixel. The pixels of the rightmost column (100 %) remain unchanged, and the resulting image effectively has a horizontal blue gradient. The key programming-related concept in this example is *for comprehension*. The result is illustrated in Figure 4.8c.

```

1  import aalto.smcl.bitmaps._

4  object FadeBlue {

6      def apply(source: Bitmap): Bitmap = {
7          val factorIncrement = 1.0 / source.widthInPixels

9          val snapshot = source.createPixelSnapshot()

11         for {
12             x <- 0 to source.widthInPixels - 1
13             factor = x * factorIncrement
14             y <- 0 to source.heightInPixels - 1
15             px = snapshot.pixel(x, y)
16         } px.blue = (factor * px.blue).toInt

18         snapshot.apply()
19     }

21 }

```

Code Example 4.4. Pixel iteration using `PixelSnapshot` and *for comprehension*. The execution result of this example is illustrated in Figure 4.8c.

Code Example 4.5 creates a horizontal color gradient by modifying both red and green components of each pixel in the same fashion as the two previous examples. The key programming-related concept in this example is *while loop*. The result is illustrated in Figure 4.8d.

```

1  import aalto.smcl.bitmaps._

4  object FadeRedGreen {

6      def apply(source: Bitmap): Bitmap = {
7          val factorIncrement = 1.0 / source.widthInPixels

9          val snapshot = source.createPixelSnapshot()

11         var x = 0
12         while (x < source.widthInPixels) {
13             val factor = x * factorIncrement

15             var y = 0
16             while (y < source.heightInPixels) {
17                 val px = snapshot.pixel(x, y)
18                 val oldRed = px.red
19                 val oldGreen = px.green

21                 px.red = (factor * oldGreen).toInt
22                 px.green = (factor * oldRed).toInt

24                 y += 1
25             }

27             x += 1
28         }

30         snapshot.apply()
31     }

33 }

```

Code Example 4.5. Pixel iteration using `PixelSnapshot` and *while loops*. The execution result of this example is illustrated in Figure 4.8d.

While in two of the previous examples the looping over pixels was explicitly written out, in the *for comprehension* example the actual looping was implicit. Code Example 4.6 resembles the latter case by hiding the looping inside of the *foreach* method, which is given a *lambda function* to execute for each of the pixels of the `PixelSnapshot`. The key programming-related concepts in this example are *foreach* method and *lambda function*. The result is illustrated in Figure 4.8e.

```

1  import aalto.smcl.bitmaps._

4  object FadeGreen {

6      def apply(source: Bitmap): Bitmap = {
7          val factorIncrement = 1.0 / source.heightInPixels

9          val snapshot = source.createPixelSnapshot()

11         snapshot foreach { px: Pixel =>
12             px.green = (px.currentYInPixels * factorIncrement * px.green).toInt
13         }

15         snapshot.apply()
16     }

18 }

```

Code Example 4.6. Pixel iteration using `PixelSnapshot` and *foreach* method with *lambda function*. The execution result of this example is illustrated in Figure 4.8e.

The last piece of program code, Code Example 4.7, is yet another way to iterate over pixels. It specifies the looping explicitly, but uses an *iterator* to fetch the pixels in a specific order. In addition, the exact continuation condition is hidden inside the iterator, which also has a utility method for indicating, if the column has changed during the previous call of the `next()` method. The key programming-related concepts in this example are *while loop* and *Iterator* pattern. The result of this program is a horizontal linear gradient, in which the amount of green decreases and the amount of blue increases when moving rightwards from the left. The result of this is illustrated in Figure 4.8f.

```
1  import aalto.smcl.bitmaps._

4  object FadeGreenBlue {

6      def apply(source: Bitmap): Bitmap = {
7          val factorDecrement = 1.0 / source.heightInPixels

9          val snapshot = source.createPixelSnapshot()

11         val it = snapshot.upwardsRightwardsIterator
12         var factor = 1.0
13         while (it.hasNext) {
14             val px = it.next()

16             if (it.columnHasChanged)
17                 factor = (factor - factorDecrement).max(0)

19             px.blue = ((1.0 - factor) * px.blue).toInt
20             px.green = (factor * px.green).toInt
21         }

23         snapshot.apply()
24     }

26 }
```

Code Example 4.7. Pixel iteration using `PixelSnapshot`, *pixel iterator*, and *while loop*. The execution result of this example is illustrated in Figure 4.8f.

After describing the research prototype’s functionality and demonstrating its usage with some examples, it is time to move on to Chapter 5, which is concerned with the implementation side of the prototype.

Chapter 5

Prototype Implementation

For this thesis, a prototype was built to evaluate possibilities of enriching Scala-based introductory programming teaching with the media computation approach. While the fourth chapter was concerned with presenting and evaluating the design and functionality of the prototype, this chapter offers insights into implementation-related aspects to provide a general-level understanding from the developer’s point of view.

The first two sections will briefly discuss the implementation of the prototype: Of the prototype’s three main components presented in § 4.4, the *Scala Media Computation Library* (SMCL) is covered in § 5.1, whereas the other two, the *Customized Scala IDE REPL* and the *SMCL External Interfaces*, are dealt with in § 5.2. The last section (§ 5.3) of this chapter discusses the project and its source resources, including essential areas for further development of the prototype.

5.1 Scala Media Computation Library

At the moment of writing, SMCL consists of over 220 Scala and Java source files, resulting over 400 compiled Java classes, interfaces and annotations¹—this section briefly introduces the most important of them. Before going into the object models, however, this section describes SMCL’s technologies, dependencies, and packages.

5.1.1 Technologies and External Dependencies

As a library based on the Scala language, SMCL (as well as the whole prototype) depends on the *Oracle Java Standard Edition Development Kit 1.7* (Figure 5.1). The version is 1.7, because it was recommended for the Scala IDE with Eclipse 4.4 at the time. The *Scala* version used by this prototype is 2.11.7, and it was selected as the latest stable version. The prototype was developed (not tested, though) with Java 1.8, too, and no complications related to it were found during the development. At the moment of writing, the next upcoming Scala release 2.12.0 requires Java 1.8 or newer, so in the near future support for version 1.7 will be dropped.

¹In fact, if counting the classes resulting from anonymous functions, the total number of classes is currently over 900. Fortunately, this seems to be changing with the upcoming Scala 2.12.0 compiler, as it does not generate separate classes for anonymous functions anymore in favor of taking advantage of Java 8 features, namely the `java.lang.invoke.LambdaMetafactory` class [Sc2120M3; LMF]. This results in both smaller application/library JAR (*Java archive*) files and fewer algorithmically-generated overlong file names to deal with under the constraints some systems set for lengths of file and path names.

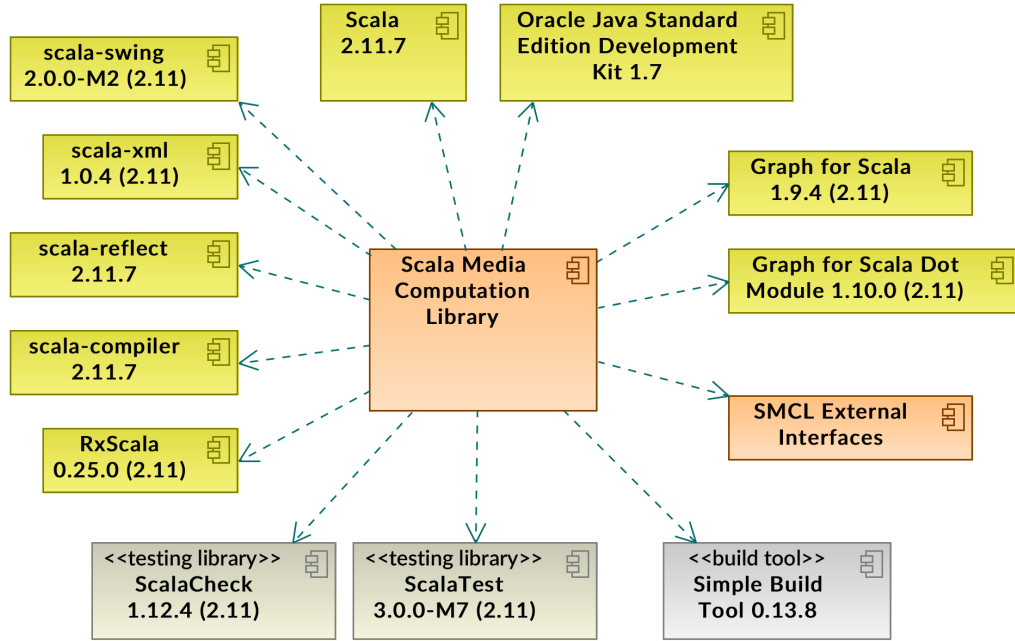


Figure 5.1. SMCL’s external dependencies. The parenthesized version numbers emphasise dependency on the Scala version used for compilation. The orange color refers to the components developed for this Thesis, whereas the yellow signifies third-party dependencies required to be available both compile and run time. The three remaining components are two testing libraries and a build tool, as denoted by their stereotypes.

In addition to Java and Scala releases, the development platform consists of several other more or less third-party components, some of which are recently separated from the class library packaged with the Scala distribution; these are the latest available stable versions supporting the Scala 2.11.x branch. The Scala-related components needed are *scala-swing 2.0.0-M2 (2.11)*, *scala-xml 1.0.4 (2.11)*, *scala-reflect 2.11.7*, and *scala-compiler 2.11.7*. Furthermore, SMCL uses *Graph for Scala 1.9.4 (2.11)* [GRS] and *Graph for Scala Dot 1.10.0 (2.11)* libraries as well as the ReactiveX API [RX] through the *RxScala 0.25.0 (2.11)* wrapper written around the RxJava library. Like previously described, the *SMCL External Interfaces* is also needed for interfacing with the prototype’s front end.

All of the components mentioned above are needed to be present at both compile and run time. Apart from them, there are currently two libraries, *ScalaCheck 1.12.4 (2.11)* [SCK] and *ScalaTest 3.0.0-M7 (2.11)* [STE], which are needed during compilation and execution of tests written for the prototype. Last, but not the least, there is the *Simple Build Tool 0.13.8* [SAX13; SBT; SF15], upon which the prototype relies its build process.

5.1.2 Packages

Figure 5.2 presents all Scala packages of SMCL, as well as their most essential mutual dependencies. SMCL’s root package is `aalto.smcl`—all other packages are contained by it. The root package contains four major packages: `infrastructure`, which contains code for platform-dependend features as well as any features not directly related to the functionality SMCL provides to its users; `common`, providing SMCL-related functionality not belonging to any other packages; `colors`, offering color-related features; and finally `bitmaps`, the root package for current main feature range, namely bitmap handling.

The `bitmaps` package contains three further packages: `metadata`, containing classes for metadata provider sources as well as metadata providers themselves; `operations`,

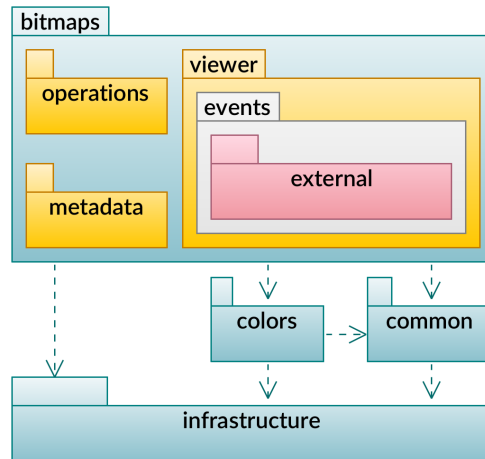


Figure 5.2. SMCL’s Scala packages and their most essential mutual dependencies. The root package containing all of these packages is `aalto.smcl`. The colors have no significance.

under which all possible built-in bitmap operations are located; and `viewer`, which holds classes related to the bitmap viewer application. The latter has a subpackage called `events`, which contains internal and external event classes for passing around information related to the bitmap viewer.

The packages directly under the root package `aalto.smcl` actualize a three-leveled layer architecture, as can be seen from Figure 5.2. The lowest layer contains only the `infrastructure` package, which does not have any dependencies to other SMCL’s packages. The next layer upwards contains packages that depend on the lowest layer but are not SMCL’s primary functionality, while the topmost layer splits the primary functionality into packages by media type. Currently there is just the `bitmaps` package, but in the future there could be also packages for sound waves, so called MIDI (*Musical Instrument Digital Interface*) music, videos, charts, bar codes, data retrieval etc.

5.1.3 Settings

The settings-related core classes are presented in Figure 5.3. A global setting store `GS` is an instance of the `Settings` class, which can include an arbitrary amount of `Setting` class instances. Each `Setting` has a type `SettingType`, a key, and a value, type of which is the specified `SettingType`. Setting keys are represented by case objects extending the

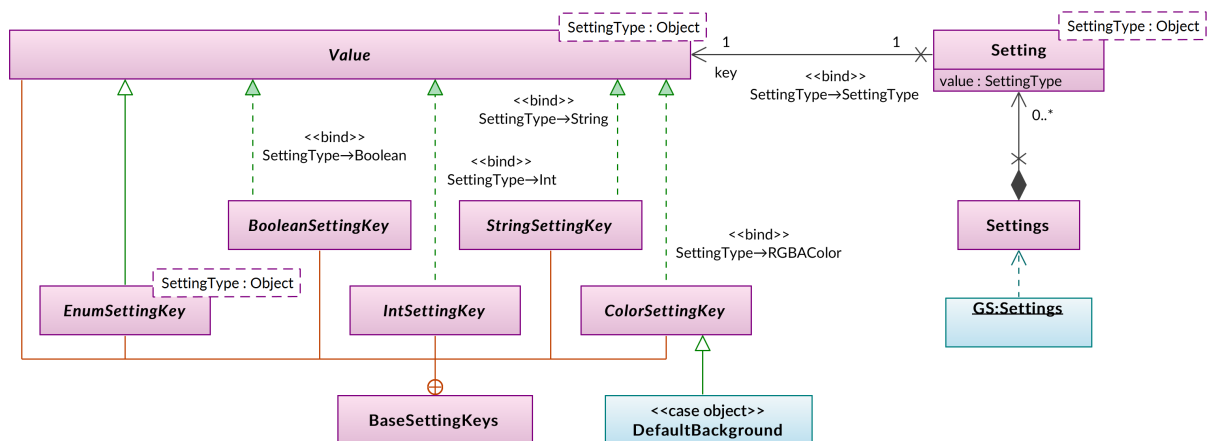


Figure 5.3. SMCL’s settings-related core Scala classes (purple) and objects (blue).

`BaseSettingKeys.Value` class, which is sealed and thus unusable outside of the `BaseSettingKeys` class. However, `BaseSettingKeys` has other inner classes extending the `Value` class and acting as base classes for all valid setting types. Like `Value`, these classes are abstract and thus not instantiable themselves. Instead, the packages defining settings declare the actual setting keys as case objects, like the `DefaultBackground` defined in the `bitmaps` package and extending the `ColorSettingKey` base class. Settings based on enumerations are a special case in that they require the enumeration to be specified as the setting type, whereas for the other setting types, the `SettingType` is permanently bound to the type the base class in question represents.

5.1.4 Colors

Figure 5.4 below contains the most important classes offering color-related functionality. Colors in SMCL are represented in the RGBA-format (red, green, blue, alpha) by the `RGBAColor` class instances, for which some extra functionality is implicitly provided by the `RichRGBAColor` class.

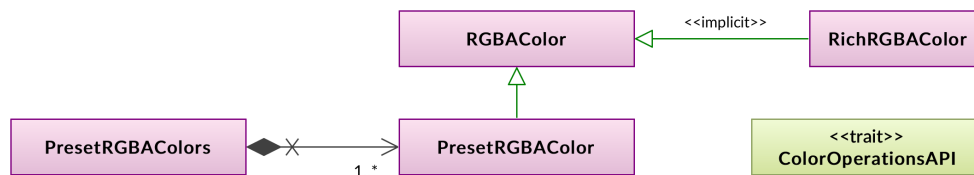


Figure 5.4. SMCL’s color-related core classes (purple) and traits (green).

Preset colors provided by SMCL are represented by the `PresetRGBAColor` class extending the `RGBAColor`. These preset colors are stored in the `sPresetRGBAColors` class. To cap it all, trait `ColorOperationsAPI` provides a bunch of color-related functionality made available by the `aalto.smcl.colors` package class via extension.

5.1.5 Bitmap Processing

The most important bitmap-related classes are presented in Figure 5.5. Currently SMCL provides only immutable bitmaps—these are implemented in the `ImmutableBitmap` case class with its companion object. Also, type `Bitmap` refers to the `ImmutableBitmap`, so that the default bitmap implementation (in case a mutable bitmap is implemented in the future) is immutable. If bitmaps are loaded from image files via the `ImmutableBitmap`’s companion object, an instance of the `BitmapLoadingResult` case class is returned, containing both the successfully loaded images and the errors (as `Throwable` instances) resulted from the loading operations.

To hold their content, each `ImmutableBitmap` has a `BitmapOperationList` case class instance containing exactly one `BufferProvider` trait implementation and an arbitrary number of `Renderable` trait implementations. Both of these traits, as well as their implementations, are located in the `aalto.smcl.bitmaps.operations` package (Figure 5.2 above). To create new bitmaps containing plain shapes (lines, rectangles, circles, ellipses, polygons etc.), one uses the functions defined in the `ShapeCreationAPI` trait, extended by the `aalto.smcl.bitmaps` package object. In addition, the independent function versions of bitmap operation calls (in contrast to the corresponding object-oriented method calls) are implemented in the `BitmapOperationAPI`, similarly extended by the package object.

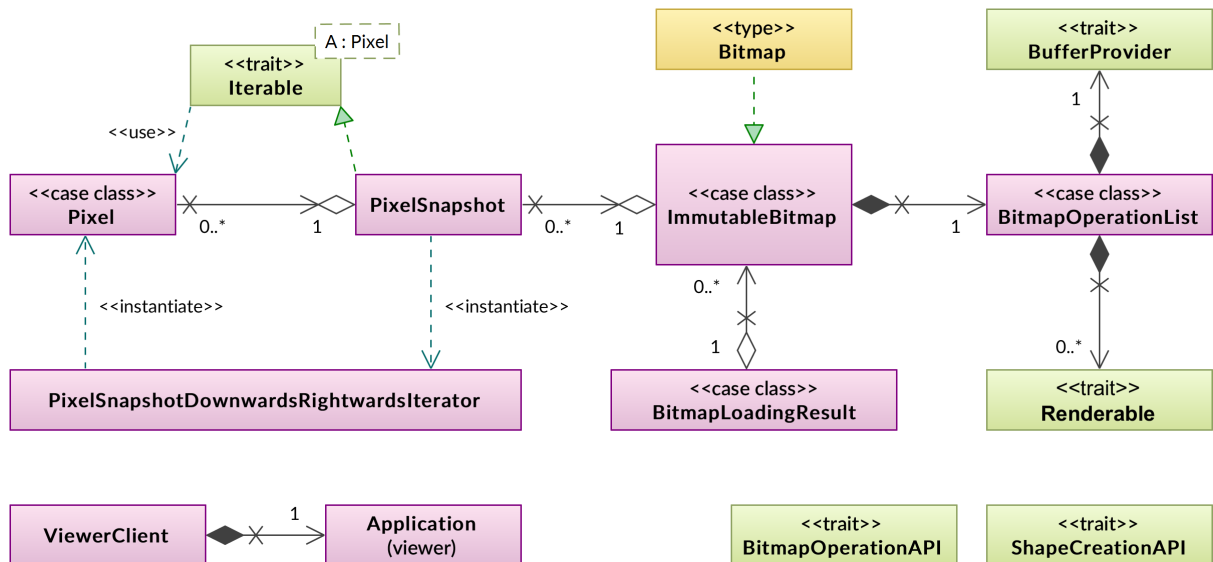


Figure 5.5. SMCL’s bitmap-related core classes (purple), traits (green), and types (yellow).

To perform custom pixel-based operations to bitmaps, one requests the bitmap for a `PixelSnapshot` class instance, which must later be applied to the same bitmap that created it. A `PixelSnapshot` instance can provide access to the actual pixels by providing instances of the `Pixel` case class either directly as return values of method calls or via an instance of a pixel iterator class, such as the `PixelSnapshotDownwardsRightwardsIterator` (there are a couple of other iterators as well).

Finally, there is the `ViewerClient` class, via which the SMCL's bitmap viewer, represented here by the `Application` class, is controlled by sending events to it. `ViewerClient`'s functionality is exposed in other contexts (such as the `display()` method/function for displaying bitmaps), and is not to be used directly by the user.

A central idea in development of SMCL was to present the bitmaps as lists and trees of operations and to render the actual bitmaps only when necessary—that is, when the bitmap is being displayed and when a custom pixel-based operation is performed for the bitmap using a `PixelFormatSnapshot` instance. This approach has several advantages: First, in the case that bitmaps are not displayed after every operation, bitmap processing is faster and memory can be saved. Second, as every bitmap knows the operation history that leads to its current content, it is possible to list those operations and even to support automatic assessment. Moreover, as the bitmap operation lists make all bitmap renditions recreatable at will, SMCL uses weak references for rendered bitmaps, so that the virtual machine is able to free up as much memory as possible.

The representation and processing of bitmaps has several problems, though. For example, as geometrical transformations are calculated for rendered bitmaps instead of the geometrical objects the bitmaps represent, performing these operations for instance for a circle reduces the quality of the image very visibly, as is demonstrated in Figure 5.6. Other bitmap interpolation methods improve the quality a little, but the degradation effect is still clearly visible, and as the figure shows, the bilinear and bicubic interpolations cause the image to become blurred and its dimensions to change. As the affine transformations are a wrapper to Java’s Abstract Windowing Toolkit (AWT) API, the interpolation methods are those offered by the `java.awt.image.AffineTransformOp` class. In SMCL, the default is *nearest neighbor*.

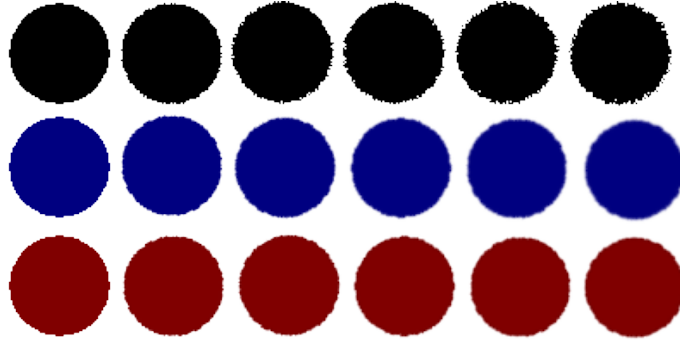


Figure 5.6. Repeated rotation’s effects to image quality. The interpolation method is *nearest neighbor* for the black circles, *bilinear* for the blue circles, and *bicubic* for the red circles. The leftmost circles are the originals, and the number of times rotated increases rightwards.

After this brief introduction to SMCL, this chapter turns attention to the Scala IDE’s customized REPL and querying metadata.

5.2 Customized Scala IDE and Metadata Query

As required by [R1](#), the primary user interface of the research prototype is the *Customized Scala IDE REPL*, which is modified to display bitmaps that illustrate the resources that are processed with SMCL. This section briefly discusses, how the bitmap display capability is implemented, as well as where the bitmaps come from and how.

Displaying Bitmaps. From the viewpoint of the *Customized Scala IDE REPL*, one important consideration is the way of presenting the media that is processed. [R2](#) requires the Scala IDE’s REPL to display embedded bitmaps, as users of modern office and graphics application suites as well as mathematical software have accustomed to expect. Unfortunately only a few of the current REPL implementations have the capability to display program code and execution results of other types than text—the rest of the REPLs are based on purely textual user interfaces (UI). For example, the Scala REPL started via the `scala` command on a command prompt clearly belongs in the group of text-only UIs, even if the traditional text mode² command line itself was emulated in a GUI, possibly in a window. Unfortunately, this category also includes Scala IDE’s REPL client, which presents a problem concerning the way in which to present the media processed with the prototype.

As told in § 4.5.2, the customized REPL of Scala IDE is capable of displaying bitmaps. However, this capability is not based on support from the control that represents the REPL’s history window. That control supports only textual content, but fortunately an approach was found to cheat it in a way that it is possible to display static bitmaps in the place of specific characters: A placeholder character can be inserted with a style that contains the bitmap to be displayed at that point and reserves enough room for it, after which a custom paint handler can check the text for the characters serving as bitmap placeholders and draw the appropriate bitmaps onto the control.

For a real application, the hack described above would naturally be highly unacceptable. For this research prototype, however, it is sufficient, and in any case, the effort required to implement a full-blown programmer’s text editor from scratch, one that has the media-handling capability, would not have fitted into the time frame of this thesis.

²*Text mode* is an operation mode of computer displays, in which the screen content is represented as a character matrix [e.g., PET00, pp. 309–313; TAN99, p. 96; MES97, pp. 1048–1095; FER94, pp. 167–211]. (For comparison, *graphic mode* enables computer software to access individual pixels on the screen).

However, this presents one significant problem to report as an answer to [RQ4](#). What is more, when content is copied to clipboard from the history list, the copied content is just text and obviously contains the placeholder characters—not the bitmaps.

Retrieving Metadata. From the viewpoint of SMCL, the bitmaps to be displayed in the REPL are considered to be metadata that describes the resources that are being processed with SMCL. The *SMCL External Interfaces* is the library that provides interfaces for querying metadata and a global registry for registering and querying classes that provide metadata for particular types of objects. The intention is that SMCL—or some other library—implements and registers classes that can provide sources for metadata (“metadata interface source provider”). These classes, in turn, are able to instantiate and provide other classes (“metadata interface source”) that both are able to provide metadata for specific types of objects and implement one or more of the query interfaces, so that the Scala IDE’s REPL can query metadata from those classes. The exact procedure is explained in a moment.

Figure 5.7 below presents two sets of resources: First, there is the content of *SMCL External Interfaces* library (no package specification). Second, as an example of an application of the traits that the *SMCL External Interfaces* provides, the figure contains the SMCL’s classes that participate in providing color-related metadata.

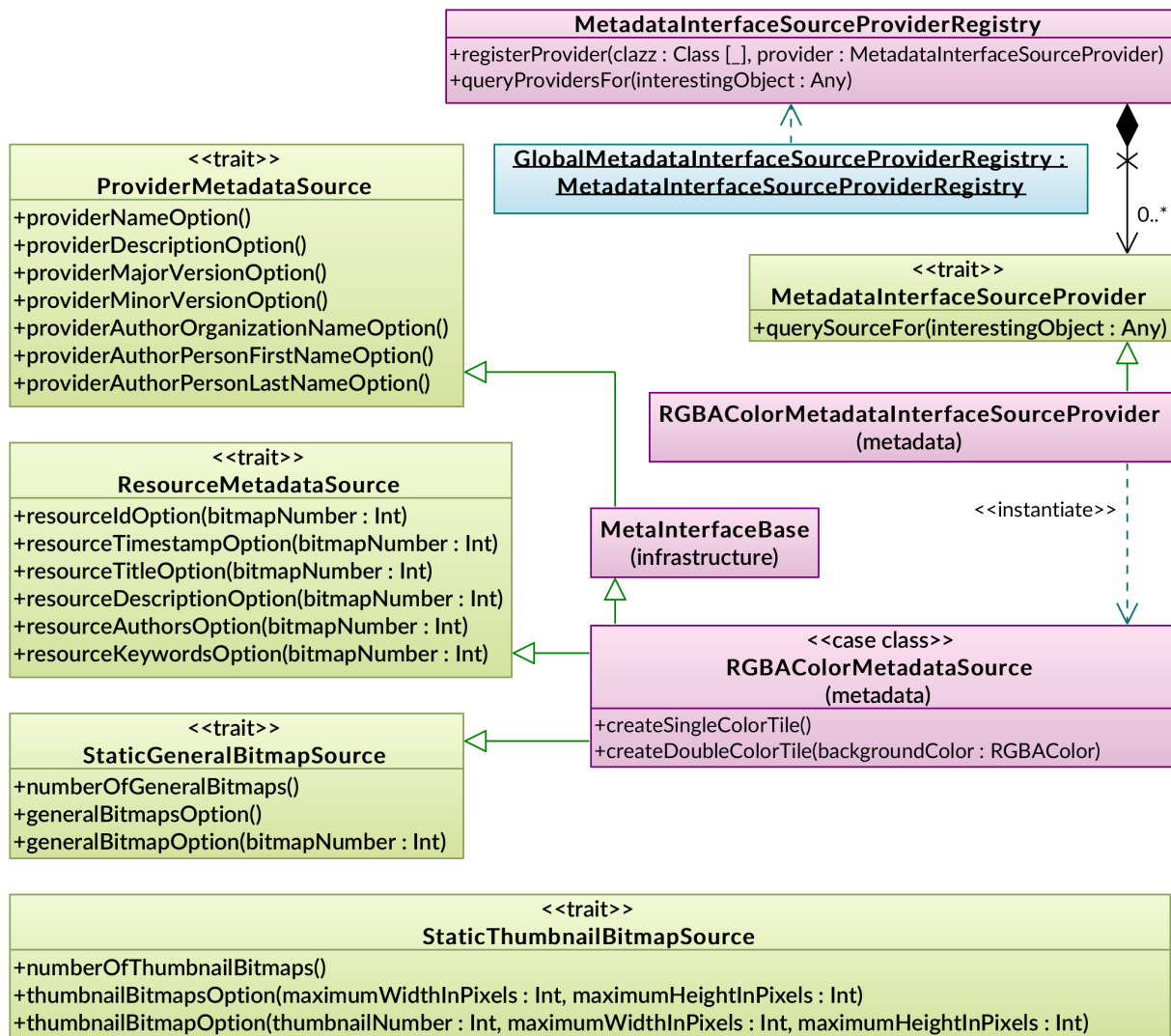


Figure 5.7. Classes (purple), traits (green), and objects (blue) from both *SMCL External Interfaces* (no package denoted) and SMCL itself (infrastructure and metadata packages).

The five traits, `ProviderMetadataSource`, `ResourceMetadataSource`, `StaticGeneralBitmapSource`, and `StaticThumbnailBitmapSource`, act as interfaces to the classes that actually provide metadata. The `MetadataInterfaceSourceProviderRegistry` and its globally available instance `GlobalMetadataInterfaceSourceProviderRegistry`, in turn, act as the first contact point when performing a metadata query. These all belong to the *SMCL External Interfaces*.

The three other classes in the figure are part of SMCL. The `MetaInterfaceBase` is intended to be a base class for all classes that provide the actual metadata. It extends the `ProviderMetadataSource` trait and describes SMCL as the metadata provider. `RGBAColorMetadataSource` is the actual class that provides color-related metadata, and it, in turn, extends the `ResourceMetadataSource` and `StaticGeneralBitmapSource` traits. `RGBAColorMetadataInterfaceSourceProvider` is the class, an instance of which SMCL registers with the `GlobalMetadataInterfaceSourceProviderRegistry` and which provides instances of the `RGBAColorMetadataSource` to describe any `RGBAColor` instances for which metadata is queried.

The procedure that the customized Scala IDE's REPL currently uses for querying metadata is presented in Algorithm 5.1 below. At the beginning, the REPL has gotten the `resultObject` from Scala interpreter as execution result and tries to present the metadata that is related to it. The procedure begins by using the global registry mentioned above to retrieve a list of classes that can provide metadata interface sources for the `resultObject` (lines 2 and 3). As the prototype provides no means to select between the metadata interface source providers returned, the prototype selects the first provider of the list (line 4), in case that at least one provider is found.

Algorithm 5.1. Metadata Search

```

1  begin
2    reg  $\leftarrow$  the GlobalMetadataInterfaceSourceProviderRegistry instance
3    providerSet  $\leftarrow$  retrieve metadata interface source providers for resultObject from reg
4    metadataSource  $\leftarrow$  first object from providerSet returning metadata source for resultObject
5    if metadataSource is instance of StaticThumbnailBitmapSource then
6      thumbnailSource  $\leftarrow$  metadataSource as StaticThumbnailBitmapSource
7      buffers  $\leftarrow$  get thumbnail bitmaps from thumbnailSource
8      add bitmaps in buffers into user interface
9    else if metadataSource is instance of StaticGeneralBitmapSource then
10     genSource  $\leftarrow$  metadataSource as StaticGeneralBitmapSource
11     buffers  $\leftarrow$  get bitmaps from genSource
12     add bitmaps in buffers into user interface

```

Primarily, the REPL tries to query thumbnail images (lines 5 to 8), but if the provider cannot provide them, the REPL tries to obtain an interface for querying full-size bitmaps (lines 9 to 12). If neither of these two interfaces are implemented, the procedure ends, because currently the prototype is not able to display any other types of information than bitmaps. The procedure for retrieving the actual bitmaps is the same for both thumbnails and full-size ones: The metadata interface source is converted into the appropriate interface, through which the bitmaps in question are then retrieved and added into the REPL's history, as was described earlier.

5.3 The Research Prototype as a Project

The project to produce the research prototype for this thesis was a very small one, but still there are many aspects that require attention especially, if this prototype is to be further developed for real use.

Dependencies. This project has several dependencies to the outside world. Obvious examples of them include third-party modules and the development tools used, but two other dependency types should be particularly taken into account. The first of them is dependency towards a single platform. As SMCL does not restrict itself to Scala’s standard library but instead both directly and indirectly uses libraries that are implemented in Java, it depends on the Java platform and does not work in Scala implementations that do not target the Java platform.

The other notable dependency is towards the Scala IDE’s developer community. As the primary user interface of the prototype is the REPL of Scala IDE, the source code of which is modified to support the prototype’s two libraries, the prototype inherently contains a customized version of the full Scala IDE distribution. In the long run, this is not a sensible approach, because both the source code of the “official” Scala IDE constantly changes and the source code of full Scala IDE is huge compared to the REPL part that is actually being used. In practice, there are two options: Either the customized REPL should become a part of the official Scala IDE distribution, or a new plug-in that contains only the customized REPL should be developed. Most likely, the first option is not realizable or even reasonable as the customized REPL supports only a single project (SMCL) that is external to the Scala IDE but should still be maintained by the Scala IDE’s developer community. This leaves the second option, with which the REPL could be developed from scratch without the dependency to Scala IDE’s developers.

Source Resources. At the moment, the source resources of the research prototype are located in two GitHub repositories (Figure 5.8) owned by the client of this thesis. The repositories are public, which means that anyone can get the source code of the prototype. However, its licensing policy is currently undecided. Furthermore, while one of the repositories is a fork of the Scala IDE and contains a branch with the *Customized Scala IDE REPL*, the second repository contains both the libraries—the *Scala Media Computation Library* and the *SMCL External Interfaces*.

It should, however, be possible to treat the libraries as independent units from each other, but as they share the same repository and a single version history, this becomes difficult. If there is to be only one Eclipse plug-in that contains the whole prototype, the *SMCL External Interfaces* could be merged to SMCL itself. Otherwise, another repository for the interface project should be established.

Build, Delivery, and Maintenance. The build process of the prototype should be further developed. SMCL and the *SMCL External Interfaces* rely on sbt [SBT] as their build tool, while the Scala IDE’s long and multiphase build is based on Apache Maven [MVN] and shell scripts. While Maven uses its own local repository system, sbt delegates module management to Apache Ivy [IVY], which in turn has another local repository system. Other reasons for improvement are that (1) the full Maven-based build of Scala IDE takes well over half an hour and seems to be error-prone, as well as that (2) at the time of writing, no one maintains the build-related shell scripts for Windows users.

The option of developing a new plug-in (discussed above) for the Scala IDE would eliminate at least the problem related to Scala IDE’s build time and errors that often seem to occur during it. If the build of the new plug-in could be sbt-based, the problem of the overlapping local repositories would be eliminated, too. If, however, Maven will be

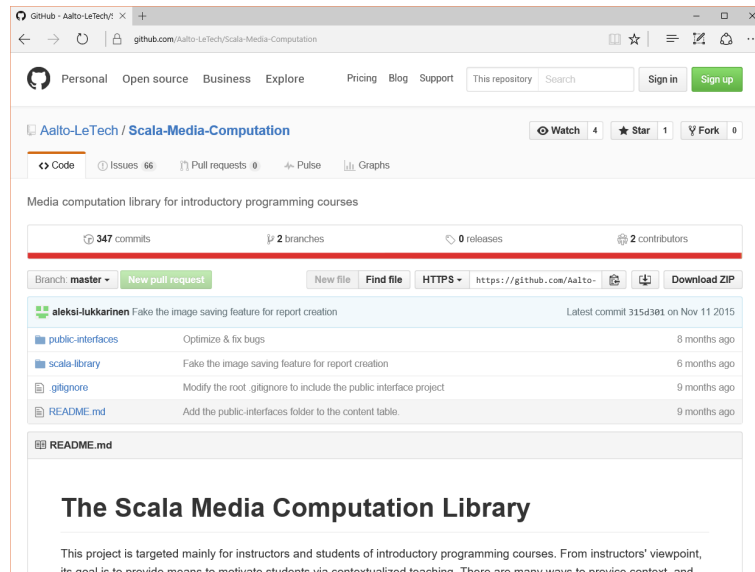


Figure 5.8. Project web page at GitHub.

used in the future, the local repositories of Maven and sbt should be combined. In any case, a continuous integration server, such as Jenkins [JNK], should be set up to take care of building the prototype, and better still, an internal module repository server (managed for instance by Nexus [NXS] or JFrog Artifactory [AFC]) could be set up to store and provision the build products and possibly also external modules.

The delivery of the executable prototype to its users, such as students, is not thought through yet. Also, the prototype's maintenance is currently open. One handy possibility for delivery would be to use Eclipse's *update site* facility (see § 6.2), if a single plug-in encapsulating the whole prototype were developed.

Implementation. While the prototype suits its role as a research aid, parts of it must be reimplemented if the intention is to give it to students to use. Most importantly, the SMCL's initialization must be redesigned, as during implementation of the prototype no reasonable approach was found for defining a global initialization to be performed when—but *before* than—any (publicly visible) member of a class library is initialized and used anywhere.

A related problem is that it seems to be challenging for a standard Scala or Java application to reliably in all cases find out all the classes that constitute a running application, which in turn makes it difficult for example to iterate over all classes of an application to determine (e.g., on the basis of annotations or interfaces) the ones that require initialization and then perform the initializing actions required by those classes. The only workaround for this would be to hard-code the required initializations into the application, which is architecturally a lousy way to implement the initialization process.

Future Development. The following list is a suggestion for most essential areas of development to be done before the prototype is ready for students to use:

- Relocate separate components into separate repositories.
- Create an Eclipse plug-in that encapsulates the three components of the prototype separately from the Scala IDE.
- Redesign SMCL's initialization procedure in a way that works reliably.
- Redesign SMCL's setting system to be user-friendly.
- Complete the bitmap viewer application or hide it from users if it is unnecessary.
- Fix any known and discovered bugs.

- Rationalize and streamline build processes.
- Adjust SMCL's API to be pedagogically reasonable (see § 6.2).
- Test and document extensively.
- Arrange means of delivery, such as an Eclipse *update site*.

The items of the following list would be nice to have but are not mandatory:

- Internal media content libraries for compositing and filtering.
- A more complete bitmap-related feature set.
- A continuous integration server for building the plug-in.
- Language levels, so that certain features could be hidden if desired.
- SMCL's settings integrated into Eclipse's settings dialog.
- Support for other types of visualizations and metadata.
- Support for other media types than bitmaps.
- A module repository server for storing and provisioning modules.
- Integration to automatic assessment servers.
- A full-scale media-aware programmer's text editor in the style of Wolfram Mathematica.
- A browser-based version of the prototype.

Bitmap-related examples of the content libraries mentioned above include Planet Cute [COO07], which is also used by the How to Design Programs' [FEL+15] teachpack `2htdp/planetcute` for DrRacket, as well as the many extensive bitmap tile sets available at the OpenGameArt.Org [OGA]. Similar resources could also be used for small-scale infographics. That way, for instance, instead of creating strings containing asterisks to represent amounts for a histogram, one could combine images for the same purpose.

The more complete bitmap-related feature set mentioned above could include items from the following list:

- Selection of files and colors using GUI-based dialogs.
- Programmatically sending bitmaps (as well as strings) to the system clipboard and loading them from it.
- Fetching bitmaps from the Internet.
- More shape primitives.
- Facilities to generate single shapes and shape lists according to specified parameters.
- Producing text with fonts.
- Using different kinds of pens and fills.
- More advanced bitmap filters to demonstrate different possibilities and to provide reference points for exercises related to implementation of those kinds of filters.
- Attaching string-based metadata to bitmaps.
- Reading and writing EXIF [CAM16], IPTC [INT16], and XMP [ADO16] metadata.

Finally, a fundamental architectural improvement would be to implement changes that are reflected by replacement of the term *bitmap* with term *image* or similar. These changes would include treating shapes as abstract geometrical objects instead of bitmap instances; keeping track of individual objects added into an image; and making the objects in an image accessible from outside of the image itself. For displaying the geometrical shapes, they could still be rendered as bitmaps by metadata provider facilities. The shapes, however, would know only their parameters as points and other types of values.

Now, after discussing the functionality and implementation of the research prototype, the only topic left in this thesis is evaluation, presented in the next chapter.

Chapter 6

Evaluation

After the discussion in chapters 2–5 concerning the research and development done for this thesis, this sixth chapter evaluates it. The evaluation starts from the research prototype (§ 6.1), continues by describing a pilot study (§ 6.2) concerning the applicability of the prototype, and concludes with the overall results of this thesis (§ 6.3), including the research questions (§ 1.3). After them, the final words of the thesis are stated in Chapter 7.

6.1 Requirements for the Research Prototype

To evaluate the research prototype developed for this thesis, its functionality is cross-checked against the requirements presented in Table 4.1. The pilot study that provides the evaluation results for the last requirement, [R9](#), is described in more detail in § 6.2.

[R1](#) : The prototype's primary user interface should be Scala IDE's REPL client.

This requirement was fully realized, as the prototype's full functionality is available when using Scala IDE's REPL.

[R2](#) : The prototype should at least embed bitmaps into Scala IDE's REPL client, as is done in the user interface of DrRacket's IDE [Rck], preferably enabling users to handle them like characters, or images in a word processing application.

The *Customized Scala IDE REPL* enables embedding bitmaps attached to the execution results displayed in the REPL's history list. However, as described in § 4.5.1, this feature is far from the full goal of [R2](#). The bitmaps and code alternate in the history list in the fashion of MATLAB's Live Editor (when results are displayed under input cells).

Furthermore, it is not possible (1) to give bitmaps as input via the command line, (2) to display code and bitmaps integrated together, nor (3) to copy or paste bitmaps either to the REPL or from it, not to mention (4) presenting media as (editable) applet-like objects, such as many word processing applications do.

[R3](#) : The prototype should be designed and packaged in a way that makes the media-related functionality usable as a class library in other REPL implementations/clients than Scala IDE's, such as a REPL being run in a console window unable to display bitmaps.

[R3](#) is fully realized, as SMCL, the library providing the actual bitmap-processing functionality, is usable also with other REPLs in environments that can take advantage of its bitmap viewer application. In addition, even without the viewer it would be suitable

for simple batch-processing applications, after adding a suitable functionality for saving bitmaps into files.

R4 : The primary programmable objects the prototype provides to its users should be immutable.

Among others, the `ImmutableBitmap` and `RGBAColor` classes are immutable, which makes this requirement fully realized.

R5 : The prototype should support both object-oriented (`object.method(parameters)`) and function-based (`function(object, parameters)`) programming.

This requirement is fully realized, as SMCL provides both object-oriented and function-based APIs for its primary programmable objects—bitmaps and colors.

R6 : It should be possible to perform bitmap-related examples presented in the Chapter 1 of *Picturing Programs* [Blo13, 9–24].

All of the examples in the first chapter can be replicated with the prototype, so this requirement is fully realized. Table 6.1 below presents examples of the counterpart operations between the first chapter of *Picturing Programs* and SMCL. It should be noted, though, that as already stated, it is not possible to combine bitmaps and program code—only to see the results of the performed operations.

R7 : It should be possible to perform bitmap-related examples presented in the Prologue of *How to Design Programs* [Fel+15].

As all of the prologue’s examples that use static bitmaps can be replicated with the prototype, so this requirement is fully realized. Table 6.2 below presents examples of the counterpart operations between the first chapter of *How to Design Programs* and SMCL. It should be noted, though, that as already stated, it is not possible to combine bitmaps and program code—only to see the results of the performed operations. Furthermore, animation is not currently supported by SMCL, either.

Table 6.1. Bitmap operations in the first chapter of *Picturing Programs* [Blo13, 9–24] with examples of their counterparts in SMCL’s object-oriented and function-based APIs.

Required Operations	Examples of Counterparts in SMCL
(flip-vertical) (flip-horizontal)	<code>flipVertically()</code> <code>flipHorizontally()</code>
(rotate-cw) (rotate-ccw) (rotate-180)	<code>rotate90DegsCw()</code> <code>rotate90DegsCcw()</code> <code>rotate180Degs()</code>
(above)	<code>appendVertically()</code> , <code>appendOnBottom()</code> , <code>:/\</code> , <code>replicateVertically()</code>
(beside)	<code>appendHorizontally()</code> , <code>appendOnLeft()</code> , <code>:/></code> , <code>replicateHorizontally()</code>
(overlay)	<code>overlayOn()</code> , <code>overlayPerAlignments()</code> , <code>:/** </code> , <code>:/*+ </code> , <code>:/*- </code>

R8 : It should be possible to perform most of the bitmap-related examples presented in *Introduction to Computing & Programming with Java* [GE05, 38–211].

As SMCL provides pixel-based access to bitmaps and as the most of the bitmap-related examples referred to by this requirement are based on looping over bitmaps' pixels, this requirement is fully realized.

In addition to the chapters 3–6 of *Introduction to Computing & Programming with Java*, SMCL also provides many of the features presented in the seventh chapter, such as drawing basic shapes and performing affine transformations. However, some of the features, such as gradient fills, pen strokes, clipping, and producing text using fonts, are not currently supported, and access to the Java's `Graphics2D` API through SMCL is denied by design.

R9 : By using the prototype, teachers should be able to design demonstrations and exercises they find pedagogically useful.

The realization of this requirement was assessed with a pilot study that is described in more detail in § 6.2 below. In summary, the teachers who participated in the study, were able to find uses for the prototype, so this requirement is deemed to be fully realized.

As a summary, apart from **R2**, all requirements set for the research prototype were successfully realized. The second requirement was only partly realized, and even the successful part, that of displaying bitmaps in Scala IDE's REPL, is based on a hack. However, for the prototype and this thesis it must be noted, that as the control used for the REPL's history window does not support bitmaps, in reality it is practically impossible to easily implement the kind of user interface desired, and implementation of a full programmer's text editor with the desired media handling capabilities requires an amount of work that was far beyond the scope of this thesis work.

In comparison with both DrRacket and the Guzdial–Ericson Multimedia Class Library, the basic functionality of SMCL is already on a good level. However, there is much to improve. For starters, SMCL does not currently provide some facilities needed to replicate all of the examples mentioned in **R8**, including dialog-based file and color selection, color distance calculation, and displaying pixel values in the bitmap viewer. Furthermore, several features of the seventh chapter of *Introduction to Computing & Programming with Java*, most importantly, producing text using fonts, are currently also missing. Moreover,

Table 6.2. Bitmap operations in the prologue of *How to Design Programs* [FEL+15, 9–24] with examples of their counterparts in SMCL's object-oriented and function-based APIs.

Required Operations	Examples of Counterparts in SMCL
(image-width) (image-height)	widthInPixels, widthRangeInPixels heightInPixels, heightRangeInPixels
(circle 10 "solid" "red")	circle(20, PresetColors('red'))
(rectangle 30 20 "outline" "blue")	Bitmap(30, 20).drawRectangle(0, 0, 29, 19, color=PresetColors('blue'))
(empty-scene 100 100)	Bitmap(100, 100)
(place-image)	overlayOn(), underlayBehind()

while the features referred to by [R6](#) and [R7](#) are implemented, the libraries in DrRacket’s distribution offer much wider capabilities than those of SMCL’s. In addition to those viewpoints, there are practically an unending list of image processing techniques that could be used as examples and for creative work when using SMCL.

What comes to the mathematics software presented earlier, the features that SMCL offers for basics-level image processing are much better than those of Maple’s and MATLAB’s, even if the functionality that *is* provided by those applications above the basics-level, is much more advanced than what SMCL currently provides. However, Mathematica’s image processing functionality is satisfactory even on basics-level, although naturally it is provided in the style of the Wolfram Language used in Mathematica.

Finally, there exist one presumably formally unpublished article [BH] about a tool that is produced in a project named SCALES and that has similar characteristics than the prototype of this thesis. The article does not discuss the tool’s functionality in detail; practically only information is that the tool offers facilities for producing two- and three-dimensional graphics as well as music, and that it utilizes other libraries to provide the functionality. As the reader knows by now, SMCL does not provide facilities for 3D graphics nor music, so in that sense the tool described in the article is a more advanced one than the prototype of this thesis. More detailed comparison concerning for example the bitmap-related functionality is not possible as no detailed information concerning the tool is provided.

6.2 Pilot Study Concerning the Research Prototype’s Applicability

As described in § 1.3 and § 1.4, due to the limited time frame of this thesis work, the applicability and results of the research prototype developed could not be studied extensively. Instead, to evaluate the fulfillment of [R9](#), the prototype was subjected to a small-scale evaluation of two teachers of the Aalto’s introductory programming courses described in § 1.2. Figure 6.1 displays the web page that was used to deliver the prototype and the related resources to the testers. Source code of SMCL was available on the SMCL’s GitHub page (Figure 5.8), and the resources delivered separately were

- a bespoke full version of Scala IDE (including the customized REPL), compiled and packaged into an *update site* to enable Eclipse to automatically download and install it
- a packaged Eclipse workspace containing the SMCL itself along with the necessary dependencies, some small test code functions, as well as sample images
- the automatically-generated documentation of SMCL.

After retrieving the resources described above and installing them, the teachers took some time to evaluate the prototype’s functionality, after which they gave feedback to the writer of this thesis. The general impression was, that the functionality provided by the prototype would benefit especially the Programming 1 course (see Figure 1.1). Possible topics for demonstrations and exercises for the first weeks of Programming 1 include nested calling of functions and methods, passing parameters, creating object instances, state versus statelessness, recursion, overloading, and repetition structures.

The improvement needs found during the pilot testing are mostly related to the SMCL’s API and include things such as naming and implementation details of some methods; presence of parameters; string-based representations of objects; Scala-specific

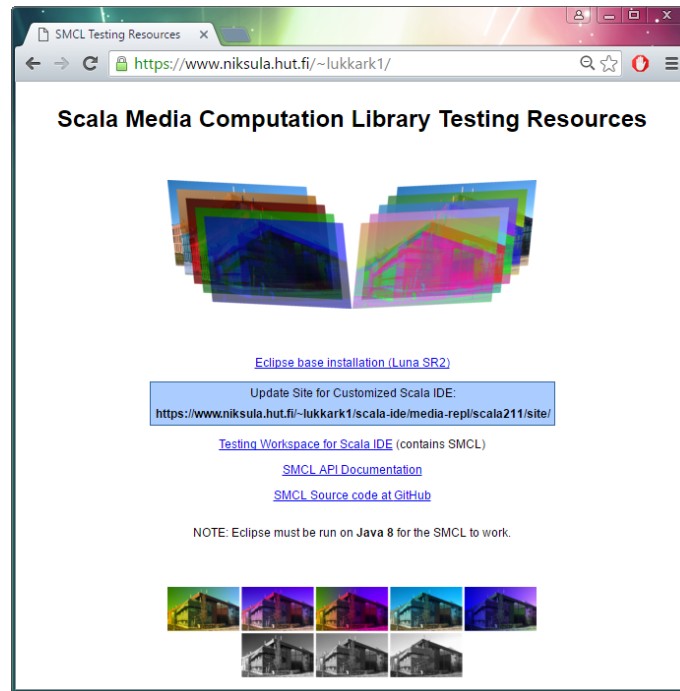


Figure 6.1. The web page distributing testing resources for the prototype.

programming style; usage of symbols; error handling (text vs. exceptions); and error message content. Furthermore, some issues with the object model were identified; these include laborious usage of preset colors, as well as the presence of the `BitmapLoadingResult` class. If the development of this prototype will be continued, all of these issues will be needed to be fixed.

6.3 Overall Results

This section discusses this thesis in its entirety, mainly based on the research questions and goals (§ 1.3) as well as the results gotten during the thesis work.

RQ1 : What kind of tool-based contextualization approaches exist for different programming languages?

Nowadays the amount of ways to contextualize teaching of introductory computer science and programming is extensive and multifaceted. This thesis presents an approximate classification (§ 2.3, Figure 2.1) of the tool-based approaches, on the basis of which the approaches can be divided into three main groups: Those that essentially combine hardware and software; those that are essentially software-based; and those that are essentially hardware-based.

The core of this thesis, Media Programming, naturally belongs under the software-based solutions, which also includes approaches such as microworlds, story-telling, games, robots, simulations, artificial intelligence, and graphical user interfaces. In some cases it may be hard to define, whether or not a single approach includes an individual set of practices. In these cases the key is to concentrate on the primary viewpoint of the context—not for instance the tools that are being used.

RQ2 : Considering the tools of **RQ1** that support Media Programming, how do their user interfaces support displaying multimedia?

As described in § 2.3, most of the current IDEs support only textual program code. Only a few IDEs have support of some degree for displaying multimedia that either is produced by executed programs or is integrated in program code; DrRacket (Figure 3.8) and Mathematica (Figure 3.9) are examples of the latter case. The classification presented in § 2.3 describes this situation more extensively, categorizing IDEs on the basis of the way they take advantage of media in their user interfaces. The sub-branches of the (2.2) *Media Programming* branch in Figure 2.1 visualizes this categorization.

RQ3 : What genres and exercise types exist in Media Programming?

As described in § 3.2, it may be difficult or impossible to exactly state what individual actions and practices Media Programming includes or excludes. However, as the classification presented in Table 3.1 illustrates, three different main genres can be identified by analyzing Guzdial’s and Ericson’s Media Computation curriculum: *Creating (Original) Information*, *Transforming Information*, and *Transforming Representation*; information-transforming includes both filtering and compositing. The table presents also exercise types using several types of media. In addition to understanding Media Programming (and Media Computation) as a contextualization approach, these can be used as ideas for new exercises. The classification as a whole can also be used to reveal new directions towards which to extend the current practices of Media Programming.

RQ4 : Considering utilization of Media Programming at Aalto University’s Department of Computer Science in Scala-based learning environments, what potential technical challenges exist?

While doing research and implementing the prototype, two primary problems were found. The first one of them is that Scala IDE’s REPL client does not support any kind of media content (see § 5.2). This means that a usable editor for Media Programming—one that hopefully resembles Mathematica’s notebooks in both structure and media usage—has to be programmed basically from scratch. There are open-source frameworks and projects that could be used to implement an editor for textual programming language, but these resources most likely do not support presenting media in any level (the research related to this was outside of the scope of this thesis).

The second problem is that during the prototype’s implementation, it was found to be challenging to create Scala-based APIs strictly of the kind described in § 4.5.2. This is because no reasonable way was found during the implementation to globally initialize the library without explicitly invoking some kind of initialization command (§ 5.3). A related problem is that of finding out which Scala/Java classes constitute a running application, which would be necessary to create an architecturally reasonable initialization procedure for the library.

RG1 : Development of a prototype that fulfills the requirements defined in the § 4.2.

To enable application of the Media Programming contextualization approach for introductory computer science and programming teaching, a prototype corresponding the requirements presented in § 4.2 was developed. It consists of three components (§ 4.4): A Scala-based class library to support bitmap-based Media Programming, a tailored version of the Scala IDE’s REPL client for displaying bitmaps in its history view (Figure 4.4), and

a set of interfaces to enable communication between these two parties. The library enables its users to create, load, filter, and composite bitmaps (§ 4.5) in an easier way than a typical image processing program code library does.

The prototype was evaluated to correspond the requirements (§ 6.1), excluding [R2](#), which was found to be too demanding for this thesis project. Moreover, the pilot testing (§ 6.2) both revealed the basic functionality to be useful for the courses it was intended for and brought out some needs for improvement before the prototype's functionality is ready for students to use.

On the basis of this chapter, the objectives of this thesis are deemed to be achieved: To research possibilities for applying Media Computation to the four courses described in § 1.2, contextualization approaches and Media Computation were explored and a research prototype was developed and evaluated to be generally useful.

The review of the literature suggests that there are no previous published efforts concerned with integrating programmatic media-related capabilities with Scala-based introductory programming teaching, excluding both Kojo (§ 3.5) and the tool from the SCALES project (§ 6.1). The research prototype developed for this thesis could be helpful in other teaching contexts similar to the four courses that were the starting point for this thesis.

However, as there is no empirical research involved with this thesis, the evaluation results of the prototype are only isolated opinions. Furthermore, the level of the writer's personal theoretical knowledge and practical skills obviously also affect to the content of this thesis. For example, when implementation of the prototype begun, the writer was not familiar with Scala and had to study it along the way.

This ends the evaluative part of this thesis. As a conclusion, the next chapter presents a summary of the content of this thesis.

Chapter 7

Conclusion

Media Programming (Chapter 3) is a subset of *Media Computation*—an approach for contextualizing teaching of programming and computer science. Contextualization itself (§ 2.2) is a well-known method for making content appear more interesting and usable for learners, and support for its applicability can be found in some theories of motivation (§ 2.1). While *Media Programming* itself is being applied and studied in several universities, there are no proper tools that enable its application to Scala-based programming education.

The starting point of this thesis were four courses (§ 1.2) that teach introductory computer science and programming using the Scala programming language. To research possibilities for applying *Media Programming* to those courses, a research prototype was implemented (Chapter 4 and Chapter 5). Moreover, to gain an extensive view of the field and to gather ideas for the prototype, the development of the prototype was preceded by a review of tool-based contextualization methods (§ 2.3) as well as existing tools that support *Media Programming* (§ 3.5).

There is a broad selection of tool-based contextualization methods—both hardware-based and software-based, as well as hybrid ones. *Media Programming* is a software-based contextualization approach; others include microworlds, story-telling, games, robots, simulations, artificial intelligence, and graphical user interfaces. Examples of purely hardware-based approaches are several board games as well as such embedded systems that can be used for programming but not be programmed by themselves. Finally, hybrid contextualization approaches include such embedded systems that themselves are programmed by the learners.

The usage of *Media Programming* or its variations as a context can be supported by several software tools, if the teaching can be realized in terms of the tool chosen to be used. Supported conventional programming languages are for example Java, Python, and for turtle graphics even Scala. In principle, several block-based languages and mathematical software are usable as well. For Scala-based programming, however, there seem to be no publicly available tools that offer simplified application programming interfaces for extensive bitmap processing, especially for pixel-based filtering.

To gain an extensive understanding of *Media Programming*, a classification (Table 3.1) was created on the basis of four books on *Media Computation*. The three main genres identified are *Creating (Original) Information*, *Transforming Information*, and *Transforming Representation*. The second genre is further divided up into *Filtering Information* and *Compositing Information*, of which the research prototype was designed to support both approaches but only with bitmaps. The prototype’s primary user interface is the same REPL (read-evaluate-print loop) client that is used in the targeted courses at Aalto—it

was modified to display bitmaps alongside execution results. The parts of the prototype that offer bitmap processing functionality to provide cognitive scaffolding for students are encapsulated into an independent class library that can be used in other Scala-based REPL implementations.

During the implementation of the prototype, two major observations were made: First, the controls used in Scala IDE's REPL support only text—not for instance bitmaps or more advanced embedded objects. Second, currently it is challenging to use Scala for creating such simplified command-line-style application programming interfaces as described in § 4.5.2. These two drawbacks should be taken into account if the research prototype is further developed. To recapitulate, the contributions of this thesis are: (1) A classification of tool-based contextualization approaches; (2) a classification of the main genres and exercise types of Media Programming; (3) the research prototype; and (4) the observations related to augmenting the REPL client of the Scala IDE's to support Media Programming.

The review of the literature suggests that no classifications of the kind of (1) or (2) have been previously reported. These classifications can help instructors and researchers to better understand the contextualization approaches available as well as to further improve their courses and teaching. The research prototype offers a tool with which to plan exercises and assess the usefulness of Media Programming in the targeted courses. It also provides a baseline for further development. Finally, the observations made during the prototype's implementation help to assess whether or not to develop the prototype further, and if so, in which direction.

A set of suggestions for more or less mandatory future work concerning the research prototype was presented in § 5.3. By implementing those suggestions, it should be possible to create a tool that, for its small part, improves the education of future software engineers.

Bibliography

- [A13] **Aalto University. 2013.** *Helsinki University of Technology*. Internet Page; Updated: Mar. 21, 2013. Available: <http://www.aalto.fi/en/about/history/tkk/>, visited on Nov. 30, 2015.
- [A15A] **Aalto University. 2015.** *Departments in brief*. Internet Page; Updated: Nov. 2015. Available: <http://sci.aalto.fi/en/departments/#CS>, visited on Nov. 30, 2015.
- [A15B] **Aalto University. 2015.** *History*. Internet Page; Updated: Nov. 2015. Available: <http://www.aalto.fi/en/about/history/>, visited on Nov. 30, 2015.
- [AAL15A] **Aalto, Maija. 2015.** *Opettaja istuu pulpettiin — Vantaalla guruoppilaat neuvovat opettajia tietotekniikassa*. Internet-sivu. Osasto: Kaupunki. Helsingin Sanomat, Sanoma Media Finland Oy. Päivitetty: joulukuu 2015. Saatavissa: <http://www.hs.fi/kaupunki/a1448949749371>, viitattu 8. joulukuuta 2015.
- [AAL15B] **Aalto, Maija. 2015.** *Sadat vanhemmat haluavat lapsensa koodikouluun — miksi? Fiksu Bella Tuomela keksi koodikoulussa, että robotit ovat aika tyhmiä*. Internet-sivu. Osasto: Kaupunki. Helsingin Sanomat, Sanoma Media Finland Oy. Päivitetty: lokakuu 2015. Saatavissa: <http://www.hs.fi/kaupunki/a1445401094647>, viitattu 26. lokakuuta 2015.
- [ABK15] **Altadmri, Amjad & Brown, Neil C.C. & Kölling, Michael.** Using BlueJ to Code Java on the Raspberry Pi. In: *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*. SIGCSE '15. New York, New York, USA: ACM, 2015. Pp. 178–178.
- [AD11] **Anderman, Eric M. & Dawson, Heather. 2011.** Learning with Motivation. In: *Handbook of Research on Learning and Instruction*. Ed. by Richard E. Mayer & Patricia A. Alexander. Educational Psychology Handbook Series. New York, New York, USA: Routledge, Taylor & Francis Group. Pp. 219–241.
- [ADO16] **Adobe Systems Incorporated. 2016.** *Adobe XMP Developer Center*. Internet Page. Available: <http://www.adobe.com/devnet/xmp.html>, visited on May 17, 2016.
- [AFC] **JFrog Ltd. 2016.** *Artifactory – The Open Source Maven Repository Manager*. Internet Site. Available: <https://www.jfrog.com/open-source/>, visited on May 8, 2016.
- [AG13] **Ademoye, Oluwakemi A. & Ghinea, Gheorghita. 2013.** Information Recall Task Impact in Olfaction-enhanced Multimedia. In: *ACM Trans. Multimedia Comput. Commun. Appl.* 9.3 (2013), 17:1–17:16.
- [AGE14A] **AgentSheets, Inc. 2014.** *AgentCubes: Creativity^{3D}*. Internet Site. Available: <http://www.agentsheets.com/agentcubes/>, visited on Mar. 5, 2016.
- [AGE14B] **AgentSheets, Inc. 2014.** *What is AgentSheets?* Internet Site. Available: <http://www.agentsheets.com/products/index.html>, visited on Mar. 5, 2016.

- [AGI16] **Agilefant Oy. 2016.** *Visibility is Agility - Agilefant*. Internet Site. Available: <http://www.agilefant.com/>, visited on Jan. 11, 2016.
- [AHO15] **Ahokas, Kari. 2015.** Pienet sormet koodaamaan. *Mikrobitti* 4 (2015), s. 42–49.
- [AK15] **Aruga, Yukika & Koike, Takafumi.** Taste Change of Soup by the Recreating of Sourness and Saltiness Using the Electrical Stimulation. In: *Proceedings of the 6th Augmented Human International Conference*. (Singapore, Singapore). AH '15. New York, New York, USA: ACM, 2015. Pp. 191–192.
- [ARD15] **Arduino LLC. 2015.** *Arduino - Home*. Internet Site. Available: <https://www.arduino.cc/>, visited on Nov. 23, 2015.
- [Ass12] **Association for Computing Machinery, Inc. 2012.** *The 2012 ACM Computing Classification System*. Internet Page; Updated: Mar. 2012. Available: <http://www.acm.org/about/class/class/2012>, visited on Dec. 31, 2015.
- [ATM] **Atom, A hackable text editor.** Internet Site. Available: <https://atom.io/>, visited on Mar. 14, 2016.
- [AUV15] **Auvinen, Tapio. 2015.** *Educational Technologies for Supporting Self-Regulated Learning in Online Learning Environments*. Article Dissertation. PDF eBook. Espoo, Finland: Department of Computer Science and Engineering, School of Science, University. 200 pp.
- [BB12] **Bartlett, Steve & Burton, Diana. 2012.** *Introduction to Education Studies*. 3rd ed. London, UK: SAGE Publications Ltd. 384 pp.
- [BBC15A] **BBC. 2015.** *Doctor Who: The Doctor and the Dalek teachers' pack*. Internet Page. Available: <http://www.bbc.co.uk/schools/0/computing/29614102>, visited on Dec. 7, 2015.
- [BBC15B] **BBC. 2015.** *The Doctor and the Dalek Game*. Internet Site. Available: <http://www.bbc.co.uk/programmes/articles/25S0qlFVD98csfFrvXcLhcr/the-doctor-and-the-dalek-game>, visited on Dec. 7, 2015.
- [BBN10] **Black, Andrew & Bruce, Kim B. & Noble, James.** Panel: Designing the Next Educational Programming Language. In: *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*. (Reno/Tahoe, Nevada, USA). OOPSLA '10. New York, New York, USA: ACM, 2010. Pp. 201–204.
- [BEL] **Bell, Joshua.** *Logo Interpreter*. Internet Site. Available: <http://www.calormen.com/jslogo/>, visited on June 2, 2015.
- [BGL12] **Bohnacker, Hartmut & Groß, Benedikt & Laub, Julia. 2012.** *Generative Design. Visualize, Program, and Create with Processing*. New York, New York, USA: Princeton Architectural Press. 474 pp.
- [BH] **Boatright, Cory D. & Howard, Brian T.** *SCALES: Learning Multimedia in a Mixed-Paradigm Language*. PDF eBook. Available: http://www.academia.edu/762462/SCALES_Learning_Multimedia_in_a_Mixed-Paradigm, visited on May 17, 2016.
- [BKK06] **Blumenfeld, Phyllis C. & Kempler, Toni M. & Krajcik, Joseph S. 2006.** Motivation and Cognitive Engagement in Learning Environments. In: *The Cambridge Handbook of the Learning Sciences*. Ed. by R. Keith Sawyer. New York, New York, USA: Cambridge University Press. Pp. 475–488.
- [BLA11] **Blake, Jonathan D.** Language Considerations in the First Year CS Curriculum. In: *Journal of Computing Sciences in Colleges*. Vol. 26. 6. USA: Consortium for Computing Sciences in Colleges, 2011. Pp. 124–129.

- [BLO13] **Bloch, Stephen. 2013.** *Picturing Programs. An Introduction to Computer Programming.* PDF eBook. 469 pp.
- [BMM06] **Brewster, Stephen & McGookin, David & Miller, Christopher.** Olfoto: Designing a Smell-based Interaction. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems.* (Montréal, Québec, Canada). CHI '06. New York, New York, USA: ACM, 2006. Pp. 653–662.
- [BRE15] **Breen, Derek. 2015.** *Scratch For Kids For Dummies.* Hoboken, New Jersey, USA: John Wiley & Sons, Inc. 384 pp.
- [BRI13] **Briggs, Jason R. 2013.** *Python for Kids. A Playful Introduction to Programming.* PDF eBook. San Francisco, California, USA: No Starch Press, Inc. 318 pp.
- [BRO09] **Brown, Andrew R. 2009.** *Making Music with Java: An introduction to computer music, Java programming, and the jMusic library.* Lulu Press, Inc. 304 pp.
- [BS15] **Bergmann, Jon & Sams, Aaron. 2015.** The Benefits And Questions Surrounding The Flipped Classroom. In: *Education Technology Solutions. For Principals / Educators / Network Administrators* 67 (2015): *Evolution Or Extinction: Why Schools Must Go Digital*, pp. 42–46.
- [BT11] **Biggs, John & Tang, Catherine. 2011.** *Teaching for Quality Learning at University.* 4th ed. Berkshire, UK: Open University Press, McGraw-Hill House. 389 pp.
- [BUR16] **Burd, Barry. 2016.** *JShell: The new Java 9 REPL.* Internet Page. Available: <http://www.theserverside.com/tip/JShell-The-new-Java-9-REPL>, visited on Mar. 16, 2016.
- [CA16] **Corno, Lyn & Anderman, Eric M., eds. 2016.** *Handbook of Educational Psychology.* 3rd ed. New York, New York, USA: Routledge, Taylor & Francis Group. 481 pp.
- [CAM16] **Camera & Imaging Products Association. 2016.** *CIPA Standards.* Internet Page. CIPA DC-010-2012 Exif 2.3 metadata for XMP. Available: http://www.cipa.jp/std/std-sec_e.html, visited on May 17, 2016.
- [CAR10] **Carl, Stephen P. 2010.** Beyond "Not-invented-here": Development Environments for a Multimedia Computation Course. In: *J. Comput. Sci. Coll.* 25.3 (2010), pp. 40–46.
- [CAR16] **Carnegie Mellon University. 2016.** *Alice.* Internet Site. Available: <http://www.alice.org/>, visited on Mar. 1, 2016.
- [CC10] **Cooper, Steve & Cunningham, Steve. 2010.** Teaching Computer Science in Context. In: *ACM Inroads* 1.1 (2010), pp. 5–8.
- [CCL] **Code Club / Home.** Internet Site. Available: <https://www.codeclub.org.uk/>, visited on Dec. 1, 2015.
- [CLI16] **Clickteam SARL. 2016.** *Clickteam Fusion 2.5.* Internet Site. Available: <http://www.clickteam.com/clickteam-fusion-2-5/>, visited on Mar. 6, 2016.
- [COD14] **Code Monkey Games, LLC. 2014.** *Code Monkey Island.* Internet Site. Available: <http://codemonkeyplanet.com/>, visited on Dec. 7, 2015.
- [COD15A] **CodeCombat Inc. 2015.** *CodeCombat.* Internet Site. Available: <https://codecombat.com/>, visited on Dec. 20, 2015.
- [COD15B] **CodeMonkey Studios, Ltd. 2015.** *Learn real coding.* Internet Site. Available: <https://www.playcodemonkey.com/>, visited on Dec. 20, 2015.

- [COD15C] **Code.org Inc. 2015.** *An Hour of Code for every student.* Internet Site. Available: [⟨https://hourofcode.com/us⟩](https://hourofcode.com/us), visited on Dec. 20, 2015.
- [COD15D] **Code.org Inc. 2015.** *Anybody can learn / Code.org.* Internet Site. Available: [⟨https://code.org/⟩](https://code.org/), visited on Dec. 20, 2015.
- [COD15E] **Code.org Inc. 2015.** *Computer Science Education Week.* Internet Site. Available: [⟨https://csedweek.org/⟩](https://csedweek.org/), visited on Dec. 20, 2015.
- [COL12] **Cole, Drusilla. 2012.** *Patterns.* London, UK: Laurence King Publishing Ltd. 320 pp.
- [COL13] **Collis, Steve. 2013.** A Beginner’s Guide to Flipped Learning. In: *Technology in Education. For Principals, Educators & Network Administrators 1* (2013): *Curriculum Shift: The Rise of Online Schooling*, pp. 44–45.
- [COO07] **Cook, Daniel. 2007.** *Danc’s Miraculously Flexible Game Prototyping Tiles.* Internet Site; Updated: May 12, 2007. Available: [⟨http://www.lostgarden.com/2007/05/dancs-miraculously-flexible-game.html⟩](http://www.lostgarden.com/2007/05/dancs-miraculously-flexible-game.html), visited on May 16, 2016.
- [CoT15] **CoToNet – Tecnologias de Informação, Unip. Lda. 2015.** *MOOC List.* Internet Page. Available: [⟨https://www.mooc-list.com/⟩](https://www.mooc-list.com/), visited on Dec. 9, 2015.
- [Cow+12] **Cowden, David & O’Neill, April & Opavsky, Erik & Ustek, Dilan & Walker, Henry M.** A C-based Introductory Course Using Robots. In: *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education.* (Raleigh, North Carolina, USA). SIGCSE ’12. New York, New York, USA: ACM, 2012. Pp. 27–32.
- [CP05] **Chastine, Jeffrey W. & Preston, Jon A.** Teaching 2D Arrays Using Real-time Video Filters. In: *Proceedings of the 6th Conference on Information Technology Education.* (Newark, New Jersey, USA). SIGITE ’05. New York, New York, USA: ACM, 2005. Pp. 133–137.
- [CUR13A] **Curran, James. 2013.** Partnering Up for Computer Science. In: *Technology in Education. For Principals, Educators & Network Administrators 2* (2013): *Coding Revolution: Rebooting Computer Science in Schools*, pp. 12–14.
- [CUR13B] **Curran, James. 2013.** Why Our Children Must Learn To Code. In: *Technology in Education. For Principals, Educators & Network Administrators 2* (2013): *Coding Revolution: Rebooting Computer Science in Schools*, pp. 6–11.
- [CUR15A] **Curran, James. 2015.** Embracing The New Technologies Curriculum. In: *Education Technology Solutions. For Principals / Educators / Network Administrators 66* (2015): *The New Digital Technologies Curriculum: Inspiring A New Generation Of Innovators*, pp. 57–58.
- [CUR15B] **Curran, James. 2015.** Getting Girls Into Coding. In: *Education Technology Solutions. For Principals / Educators / Network Administrators 66* (2015): *The New Digital Technologies Curriculum: Inspiring A New Generation Of Innovators*, pp. 12–14.
- [DAN+12] **Dann, Wanda & Cosgrove, Dennis & Slater, Don & Culyba, Dave & Cooper, Steve.** Mediated Transfer: Alice 3 to Java. In: *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education.* SIGCSE ’12. New York, New York, USA: ACM, 2012. Pp. 141–146.
- [DCE10] **Dann, Wanda & Cooper, Stephen & Ericson, Barbara. 2010.** *Exploring Wonderland. Java Programming Using Alice and Media Computation.* Upper Saddle River, New Jersey, USA: Pearson Education, Inc. 652 pp.

- [DEN14] **Denning, Peter. 2014.** Interview with Mark Guzdial, Georgia Institute of Technology: Computing As Creation. In: *Ubiquity 2014* (2014), 1:1–1:7.
- [DIC15] **Dickins, Rosie. 2015.** *Lift-the-Flap Computers and Coding*. Usborne Books. 16 pp.
- [DKP14A] **DK Publishing. 2014.** *Computer Coding. An Introduction to Computer Programming*. Series: DK Workbooks. London, UK: Dorling Kindersley Limited. 40 pp.
- [DKP14B] **DK Publishing. 2014.** *Help Your Kids with Computer Coding. A Unique Step-by-Step Visual Guide, from Binary Code to Building Games*. London, UK: Dorling Kindersley Limited. 224 pp.
- [DOR14] **Dorn, Brian. 2014.** *Welcome to Jeroo!* Internet Site. Available: <http://home.cc.gatech.edu/dorn/jeroo>, visited on Mar. 6, 2016.
- [DOU14] **Douglas, Tim. 2014.** Learning Personalized. Putting the student-centric approach in focus. In: *entrsekt. Where learning, technology and community meet* (2014), pp. 26–31.
- [DOU15] **Douglas, Tim. 2015.** <cracking the code>. Computer science, coding becoming worldwide focus for new curriculum. In: *entrsekt. Where learning, technology and community meet* (2015), pp. 16–21.
- [DPW] **Rickitt Educational Media, Ltd. 2014.** *Dragon Pathways*. Internet Site. Available: <http://www.r-e-m.co.uk/logo/?Titleno=29300>, visited on Mar. 5, 2016.
- [DRE14A] **Dredge, Stuart. 2014.** *Hello Ruby kids' coding book raises \$185k (and counting) on Kickstarter*. Internet Page. Guardian News and Media Limited. Updated: Aug. 2014. Available: <http://www.theguardian.com/technology/2014/jan/27/hello-ruby-kids-coding-book-kickstarter>, visited on June 24, 2015.
- [DRE14B] **Dredge, Stuart. 2014.** *Kids coding at school: 'When you learn computing, you're thinking about thinking'*. Internet Page. Guardian News and Media Limited. Updated: Sept. 2014. Available: <http://www.theguardian.com/technology/2014/sep/22/computing-bcs-uk-computing-curriculum>, visited on Dec. 8, 2015.
- [DRE14C] **Dredge, Stuart. 2014.** *Minecraft Add-On LearnToMod Aims to Teach Children Coding Skills*. Internet Page. Guardian News and Media Limited. Updated: Aug. 2014. Available: <http://www.theguardian.com/technology/2014/aug/19/minecraft-learntomod-children-coding-programming>, visited on Dec. 1, 2015.
- [DRE15A] **Dredge, Stuart. 2015.** *Can programmable robots Dot and Dash teach your kids to code?* Internet Page. Guardian News and Media Limited. Updated: Sept. 2015. Available: <http://www.theguardian.com/technology/2015/sep/07/robots-teach-kids-to-code-dot-dash>, visited on Dec. 8, 2015.
- [DRE15B] **Dredge, Stuart. 2015.** *Classic children's books on coding reprogrammed for a new generation*. Internet Page. Guardian News and Media Limited. Updated: Oct. 1, 2015. Available: <http://www.theguardian.com/technology/2015/oct/01/usborne-books-teach-children-code-lisa-watts>, visited on Dec. 8, 2015.
- [DRE15C] **Dredge, Stuart. 2015.** *Coding at school: A Parent's Guide to England's New Computing Curriculum*. Internet Page. Guardian News and Media Limited. Updated: Sept. 2015. Available: <http://www.theguardian.com/technology/2014/sep/04/coding-school-computing-children-programming>, visited on Dec. 1, 2015.

- [DRE15D] **Dredge, Stuart. 2015.** *Forcing a generation to code is unprecedented, says Codecademy chief.* Internet Page. Guardian News and Media Limited. Updated: Sept. 2015. Available: <http://www.theguardian.com/technology/2014/sep/05/codecademy-coding-schools-education-apps>, visited on Dec. 1, 2015.
- [DRJ] **DrJava.** Internet Site. Available: <http://www.drjava.org/>, visited on Mar. 14, 2016.
- [DSM15] **Dickins, Rosie & Stowell, Louie & Melmoth, Jonathan. 2015.** *Coding for Beginners using Scratch.* Usborne Books. 96 pp.
- [EBK12] **Ehmann, Sven & Bohl, Stephan & Klanten, Robert, eds. 2012.** *Cause and Effect. Visualizing Sustainability.* Berlin, Germany: Die Gestalten Verlag GmbH & Co. KG. 240 pp.
- [ECL15] **Eclipse Foundation. 2015.** *Eclipse - The Eclipse Foundation open source community website.* Internet Site. Available: <https://eclipse.org/home/index.php>, visited on Nov. 23, 2015.
- [EFF00] **Efford, Nick. 2000.** *Digital Image Processing: A practical introduction using Java.* Harlow, Essex, England: Jones & Bartlett Learning. 372 pp.
- [ELI15] **Elisa Oyj. 2015.** *Tervetuloa digikouluun!* Internet-sivu. Saatavissa: <https://elisa.fi/digikoulu/>, viitattu 20. joulukuuta 2015.
- [EMB16] **Embarcadero Technologies, Inc. 2016.** *Home - Embarcadero Website.* Internet Site. Available: <http://www.embarcadero.com/>, visited on Mar. 14, 2016.
- [ERI+15] **Ericson, Barbara & Guzdial, Mark & Morrison, Briana & Parker, Miranda & Moldavan, Matthew & Surasani, Lekha. 2015.** An eBook for Teachers Learning CS Principles. In: *ACM Inroads* 6.4 (2015), pp. 84–86.
- [EST14] **Estep, Linda A. 2014.** Gathering STEAM. Integration of arts into STEM inspires new mindset. In: *entrsekt. Where learning, technology and community meet* (2014), pp. 16–21.
- [ETY] **Squeakland: Home of Squeak Etoys.** Internet Site. Available: <http://www.squeakland.org/>, visited on Mar. 6, 2016.
- [FAC15] **Faculty of Engineering, Lund University. 2015.** *Scala will be a beginner programming language at LTH.* Internet Page; Updated: Nov. 2015. Available: <https://www.lth.se/english/about-lth/news/news/article/scala-will-be-the-beginner-programming-language-of/>, visited on Dec. 30, 2015.
- [FAW12] **Fawcett-Tang, Roger. 2012.** *Numbers in Graphic Design.* London, UK: Laurence King Publishing Ltd. 320 pp.
- [FEL+15] **Felleisen, Matthias & Findler, Robert Bruce & Flatt, Matthew & Krishnamurthi, Shriram. 2015.** *How to Design Programs.* Internet Site. 2nd Edition. Stable Release, Version 6.2.900.6. Updated: Aug. 6, 2015. Available: <http://www.ccs.neu.edu/home/matthias/HtDP2e/>, visited on Nov. 16, 2015.
- [FER94] **Ferraro, Richard F. 1994.** *Programmer's Guide to the EGA, VGA, and Super VGA Cards.* 3rd ed. 2nd printing, June 1995. Reading, Massachusetts, USA: Addison-Wesley Publishing Company, Inc. 1600 pp.
- [FJ14] **Friedland, Gerald & Jain, Ramesh. 2014.** *Multimedia Computing.* Kindle eBook. New York, New York, USA: Cambridge University Press.

- [FMV15] **Freeman, Jason & Magerko, Brian & Verdin, Regis.** EarSketch: A Web-based Environment for Teaching Introductory Computer Science Through Music Remixing. In: *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*. (Kansas City, Missouri, USA). SIGCSE '15. New York, New York, USA: ACM, 2015. Pp. 5–5.
- [Foo] **The Foos.** Internet Page. CodeSpark, Inc. Updated: 2015. Available: <http://www.thefoos.com/>, visited on Dec. 20, 2015.
- [Fox13] **Fox, Armando. 2013.** From MOOCs to SPOCs. In: *Commun. ACM* 56.12 (2013), pp. 38–40.
- [FP04A] **Fincher, Sally & Petre, Marian, eds. 2004.** *Computer Science Education Research*. Lisse, Netherlands: RoutledgeFalmer, Taylor & Francis The Netherlands. 239 pp.
- [FP04B] **Fincher, Sally & Petre, Marian. 2004.** Part One: The Field and the Endeavor. In: *Computer Science Education Research*. Ed. by Sally Fincher & Marian Petre. Lisse, Netherlands: RoutledgeFalmer, Taylor & Francis The Netherlands. Pp. 1–81.
- [FRE+10] **Freudenthal, Eric Andrew & Roy, Mary K. & Ogrey, Alexandria Nicole & Magoc, Tanja & Siegel, Alan.** MPCT: Media Propelled Computational Thinking. In: *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*. (Milwaukee, Wisconsin, USA). SIGCSE '10. New York, New York, USA: ACM, 2010. Pp. 37–41.
- [FRE+14] **Freeman, Jason & Magerko, Brian & McKlin, Tom & Reilly, Mike & Permar, Justin & Summers, Cameron & Fruchter, Eric.** Engaging Underrepresented Groups in High School Introductory Computing Through Computational Remixing with EarSketch. In: *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*. (Atlanta, Georgia, USA). SIGCSE '14. New York, New York, USA: ACM, 2014. Pp. 85–90.
- [FRE14] **Free Software Foundation, Inc. 2014.** *Overview of the GNU System*. Internet Page. Available: <http://www.gnu.org/gnu/gnu-history.html>, visited on Dec. 31, 2015.
- [FRE15] **Free Software Foundation, Inc. 2015.** *GNU Emacs*. Internet Site. Available: <https://www.gnu.org/software/emacs/>, visited on Mar. 14, 2016.
- [GA10] **Ghinea, Gheorghita & Ademoye, Oluwakemi.** A User Perspective of Olfaction-enhanced Mulsemmedia. In: *Proceedings of the International Conference on Management of Emergent Digital EcoSystems*. (Bangkok, Thailand). MEDES '10. New York, New York, USA: ACM, 2010. Pp. 277–280.
- [GA12] **Ghinea, Gheorghita & Ademoye, Oluwakemi. 2012.** The Sweet Smell of Success: Enhancing Multimedia Applications with Olfaction. In: *ACM Trans. Multimedia Comput. Commun. Appl.* 8.1 (2012), 2:1–2:17.
- [GAM+95] **Gamma, Erich & Helm, Richard & Johnson, Ralph & Vlissides, John. 1995.** *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Upper Saddle River, New Jersey, USA: Pearson Education, Inc. 395 pp.
- [GAM14] **GameSalad, Inc. 2014.** *Want to make games?* Internet Site. Available: <https://gamesalad.com/>, visited on Mar. 6, 2016.
- [GE05] **Guzdial, Mark J. & Ericson, Barbara. 2005.** *Introduction to Computing & Programming with Java. A Multimedia Approach*. Upper Saddle River, New Jersey, USA: Pearson Education, Inc. 558 pp.

- [GE11] **Guzdial, Mark J. & Ericson, Barbara. 2011.** *Problem Solving with Data Structures Using Java. A Multimedia Approach.* Upper Saddle River, New Jersey, USA: Pearson Education, Inc. 507 pp.
- [GE13] **Guzdial, Mark J. & Ericson, Barbara. 2013.** *Introduction to Computing and Programming in Python. A Multimedia Approach.* Kindle eBook. Third International. Essex, UK: Pearson Education Limited.
- [GF05] **Guzdial, Mark & Forte, Andrea.** Design Process for a Non-majors Computing Course. In: *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education.* (St. Louis, Missouri, USA). SIGCSE '05. New York, New York, USA: ACM, 2005. Pp. 361–365.
- [GFT] **Greenfoot: Teach & Learn Java Programming.** Internet Site. Available: <http://www.greenfoot.org/>, visited on Mar. 6, 2016.
- [GIR15] **Girls Who Code, Inc. 2015.** *Girls Who Code.* Internet Site. Available: <http://girlswhocode.com/>, visited on Dec. 20, 2015.
- [GKX12] **Greenberg, Ira & Kumar, Deepak & Xu, Dianna.** Creative Coding and Visual Portfolios for CS1. In: *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education.* (Raleigh, North Carolina, USA). SIGCSE '12. New York, New York, USA: ACM, 2012. Pp. 247–252.
- [GMP] **GIMP - GNU Image Manipulation Program.** Internet Site; Updated: 2016. Available: <https://www.gimp.org/>, visited on Apr. 17, 2016.
- [Goo15] **Google Developers. 2015.** *Blockly.* Internet Site. Available: <https://developers.google.com/blockly/>, visited on Mar. 6, 2016.
- [GRS] **Graph for Scala – Home.** Internet Site; Updated: 2015. Available: <http://www.scala-graph.org/>, visited on Oct. 26, 2015.
- [GS02] **Guzdial, Mark & Soloway, Elliot. 2002.** Teaching the Nintendo Generation to Program. In: *Commun. ACM* 45.4 (2002), pp. 17–21.
- [GUZ+10] **Guzdial, Mark & Ranum, David & Miller, Brad & Simon, Beth & Ericson, Barbara & Rebelsky, Samuel A. & Davis, Janet & Deepak, Kumar & Blank, Doug.** Variations on a Theme: Role of Media in Motivating Computing Education. In: *Proceedings of the 41st ACM Technical Symposium on Computer Science Education.* (Milwaukee, Wisconsin, USA). SIGCSE '10. New York, New York, USA: ACM, 2010. Pp. 66–67.
- [GUZ03] **Guzdial, Mark.** A Media Computation Course for Non-majors. In: *Proceedings of the 8th Annual Conference on Innovation and Technology in Computer Science Education.* (Thessaloniki, Greece). ITiCSE '03. New York, New York, USA: ACM, 2003. Pp. 104–108.
- [GUZ04] **Guzdial, Mark. 2004.** Programming Environments for Novices. In: *Computer Science Education Research.* Ed. by Sally Fincher & Marian Petre. Lisse, Netherlands: RoutledgeFalmer, Taylor & Francis The Netherlands. Pp. 127–154.
- [GUZ09A] **Guzdial, Mark. 2009.** *Defining Media Computation: Shifting levels of abstraction for fun.* Internet Page; Updated: Aug. 2009. Available: <https://computinged.wordpress.com/2009/08/10/defining-media-computation-shifting-levels-of-abstraction-for-fun/>, visited on Feb. 2, 2016.
- [GUZ09B] **Guzdial, Mark. 2009.** *Options for a totally blind CS1 student.* Internet Page; Updated: Dec. 2009. Available: <https://computinged.wordpress.com/2009/12/18/options-for-a-totally-blind-cs1-student/>, visited on Mar. 30, 2016.

- [Guz10] **Guzdial, Mark. 2010.** Does Contextualized Computing Education Help? In: *ACM Inroads* 1.4 (2010), pp. 4–6.
- [Guz13] **Guzdial, Mark.** Exploring Hypotheses About Media Computation. In: *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research*. (San Diego, California, USA). ICER '13. New York, New York, USA: ACM, 2013. Pp. 19–26.
- [Guz14] **Guzdial, Mark. 2014.** Why the U.S. Is Not Ready for Mandatory CS Education. In: *Commun. ACM* 57.8 (2014), pp. 8–9.
- [Guz15] **Guzdial, Mark. 2015.** *Learner-Centered Design of Computing Education. Research on Computing for Everyone*. Series: Synthesis Lectures on Human-Centered Informatics. Lecture #33. London, UK: Morgan & Claypool Publishers. 165 pp.
- [Guz16] **Guzdial, Mark. 2016.** *How to Teach Computer Science with Media Computation*. Internet Page; Updated: May 2016. Available: <https://computinged.wordpress.com/2015/05/13/how-to-teach-computer-science-with-media-computation/>, visited on Mar. 30, 2016.
- [HAA15] **Haakana, Kari. 2015.** Älä opettele ohjelmoimaan. *Mikrobitti* 2 (2015), s. 58.
- [HAR14] **Hartig, Ossian. 2014.** *Rehtori kokeili ohjelmointia: “Kansalaistaito siinä missä moni muukin”*. Internet-sivu. Tivi, Talentum. Päivitetty: joulukuu 2014. Saatavissa: <http://www.tivi.fi/Uutiset/2014-12-12/Rehtori-kokeili-ohjelmointia-Kansalaistaito-siin%C3%A4-miss%C3%A4-moni-muukin-3151385.html>, viitattu 3. kesäkuuta 2015.
- [HEL15A] **Hello Ruby Oy. 2015.** *Home / Hello Ruby*. Internet Site. Available: <http://www.helloruby.com/>, visited on Dec. 7, 2015.
- [HEL15B] **Helsingin yliopisto. 2015.** *Linkki: Tietojenkäsittelyn resurssikeskus*. Internet-sivusto. Saatavissa: <http://linkki.cs.helsinki.fi/>, viitattu 1. joulukuuta 2015.
- [HEL15C] **Helsingin yliopisto, Tietojenkäsittelytieteen laitos. 2015.** *Korkeakoulu kotonasi*. Internet-sivu. Saatavissa: <http://mooc.fi/>, viitattu 9. joulukuuta 2015.
- [HEL16] **Helsingin kaupunki. 2016.** *Kymmenessä koulussa kokeillaan robottien käyttöä opetuksessa*. Internet-sivu; Päivitetty: 29. helmikuuta 2016. Saatavissa: <http://www.hel.fi/www/uutiset/fi/opetusvirasto/zora-koulussa>, viitattu 4. maaliskuuta 2016.
- [HER15A] **Herala, Antti. 2015.** *Developing an Introductory Object-Oriented Programming Course*. Master’s Thesis. PDF eBook. Lappeenranta, Finland: Degree Program in Computer Science, School of Business and Management, Lappeenranta University of Technology. 80 pp.
- [HER15B] **Hernandez, Christian. 2015.** *Every child should learn to program, but not necessarily how to code*. Internet Page. Guardian News and Media Limited. Updated: Oct. 2015. Available: <http://www.theguardian.com/media-network/media-network-blog/2014/oct/20/teaching-children-programme-code-technology-creativity>, visited on Dec. 8, 2015.
- [HER16] **Her Interactive, Inc. 2016.** *Nancy Drew: Codes & Clues*. Internet Site. Available: <http://www.herinteractive.com/nancy-drew-codes-clues/>, visited on Feb. 7, 2016.

- [HIN15] **Hinsliff, Gaby. 2015.** *Should kids learn to code?* Internet Page. Guardian News and Media Limited. Updated: Dec. 2015. Available: <http://www.theguardian.com/news/2015/dec/03/should-kids-learn-code>, visited on Dec. 7, 2015.
- [HM14] **Herrera, Nicolas S. & McMahan, Ryan P.** Development of a Simple and Low-Cost Olfactory Display for Immersive Media Experiences. In: *Proceedings of the 2Nd ACM International Workshop on Immersive Media Experiences*. (Orlando, Florida, USA). ImmersiveMe '14. New York, New York, USA: ACM, 2014. Pp. 1–6.
- [Ho16] **Ho, Don. 2016.** *Notepad++ Home*. Internet Site. Available: <https://notepad-plus-plus.org/>, visited on Mar. 14, 2016.
- [HRM] **Tomorrow Corporation.** *Tomorrow Corporation : Human Resource Machine*. Internet Site. Available: <https://tomorrowcorporation.com/humanresourcemachine>, visited on Apr. 2, 2015.
- [HSC] **Hopscotch Technologies, Inc.** *Hopscotch: Coding Made Awesome*. Internet Site. Available: <https://www.gethopscotch.com/>, visited on Mar. 6, 2016.
- [HUM91] **Humphrey, Watts S. 1991.** Software and the Factory Paradigm. In: *Software Engineering Journal* 6.5 (1991), pp. 370–376.
- [HYA06] **Hyacinthe, Berg P.** Apparatus and Methods for Production of Printed Aromatic and Gustative Information. In: *Proceedings of the 6th ACM/IEEE-CS Joint Conference on Digital Libraries*. (Chapel Hill, North-Carolina, USA). JCDL '06. New York, New York, USA: ACM, 2006. Pp. 365–365.
- [ILL09] **Illeris, Knud, ed. 2009.** *Contemporary Theories of Learning. Learning Theorists... in Their Own Words*. Abingdon, Oxfordshire, UK: Routledge, Taylor & Francis Group. 235 pp.
- [INN15] **Innokas Network. 2015.** *Innokas!* Internet-sivusto. Saatavissa: <http://www.innokas.fi/>, viitattu 8. joulukuuta 2015.
- [INT15] **International Computing Education Research Conference. 2015.** *About*. Internet Page. Available: <http://icer.hosting.acm.org/general-info/about/>, visited on Jan. 11, 2016.
- [INT16] **International Press Telecommunications Council. 2016.** *Photo Metadata*. Internet Site. Available: <https://iptc.org/standards/photo-metadata/>, visited on May 17, 2016.
- [ISH+12] **Ishibashi, Yutaka & Hoshino, Sosuke & Zeng, Qi & Fukushima, Norishige & Sugawara, Shinji.** QoE Assessment of Fairness Between Players in Networked Game with Olfaction. In: *Proceedings of the 11th Annual Workshop on Network and Systems Support for Games*. (Venice, Italy). NetGames '12. Piscataway, New Jersey, USA: IEEE Press, 2012. 12:1–12:2.
- [IVA+15] **Ivanoviü, Mirjana & Budimac, Zoran & Radovanoviü, Miloš & Saviü, Miloš.** Does the Choice of the First Programming Language Influence Students' Grades? In: *Proceedings of the 16th International Conference on Computer Systems and Technologies*. (Dublin, Ireland). Ed. by Boris Rachev & Angel Smrikarov. CompSysTech '15. New York, New York, USA: ACM, 2015. Pp. 305–312.
- [IVY] **The Apache Software Foundation. 2014.** *Ivy: The agile dependency manager*. Internet Site. Available: <http://ant.apache.org/ivy/>, visited on May 8, 2016.

- [JES] *gatech-csl/jes*. Internet Site; Updated: 2015. Available: [⟨https://github.com/gatech-csl/jes⟩](https://github.com/gatech-csl/jes), visited on Mar. 14, 2016.
- [JET15] **JetBrains, s.r.o. 2015.** *IntelliJ IDEA — The Most Intelligent Java IDE*. Internet Site. Available: [⟨https://www.jetbrains.com/idea/⟩](https://www.jetbrains.com/idea/), visited on Nov. 23, 2015.
- [JHN] *Jython: Python for the Java Platform*. Internet Site. Available: [⟨http://www.jython.org/⟩](http://www.jython.org/), visited on Apr. 9, 2016.
- [JNK] *Jenkins: Build great things at any scale*. Internet Site; Updated: 2016. Available: [⟨https://jenkins.io/⟩](https://jenkins.io/), visited on May 8, 2016.
- [JOH14] **Johnston, David. 2014.** Brisbane college successfully introduces true BYOD. In: *Technology in Education. For Principals, Educators & Network Administrators 3* (2014): *SOLE: Self-Organized Learning Environments Alive and Well in Australian Schools*, p. 35.
- [KA] *Koodiaapinen MOOC*. Internet-sivusto; Päivitetty: 2015. Saatavissa: [⟨http://koodiaapinen.fi/⟩](http://koodiaapinen.fi/), viitattu 8. joulukuuta 2015.
- [KAF+13] **Kafai, Yasmin B. & Searle, Kristin & Kaplan, Eliot & Fields, Deborah & Lee, Eunyoung & Lui, Debora.** Cupcake Cushions, Scooby Doo Shirts, and Soft Boomboxes: E-Textiles in High School to Promote Computational Concepts, Practices, and Perceptions. In: *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*. (Denver, Colorado, USA). SIGCSE '13. New York, New York, USA: ACM, 2013. Pp. 311–316.
- [KAF+14] **Kafai, Yasmin & Searle, Kristin & Martinez, Cristóbal & Brayboy, Bryan.** Ethnocomputing with Electronic Textiles: Culturally Responsive Open Design to Broaden Participation in Computing in American Indian Youth and Communities. In: *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*. (Atlanta, Georgia, USA). SIGCSE '14. New York, New York, USA: ACM, 2014. Pp. 241–246.
- [KB12] **Kölling, Michael & Brown, Neil.** Teaching with Greenfoot and the Kinect: A Novel Way to Engage Beginners (Abstract Only). In: *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*. (Raleigh, North Carolina, USA). SIGCSE '12. New York, New York, USA: ACM, 2012. Pp. 659–659.
- [KHA15A] **Khan Academy, Inc. 2015.** *Khan Academy*. Internet Site. Available: [⟨http://www.khanacademy.org/⟩](http://www.khanacademy.org/), visited on Dec. 20, 2015.
- [KHA15B] **Khan Academy, Inc. 2015.** *Microsoft YouthSpark Hub / Student Career / Computer Science Skills*. Internet Site. Available: [⟨http://www.microsoft.com/about/corporatecitizenship/en-us/youthspark/youthsparkhub/⟩](http://www.microsoft.com/about/corporatecitizenship/en-us/youthspark/youthsparkhub/), visited on Dec. 20, 2015.
- [KK13] **Korkmaz, Sedat & Korkmaz, Şule Çelik. 2013.** Contextualization or De-contextualization: Student Teachers' Perceptions about Teaching a Language in Context. In: *Procedia - Social and Behavioral Sciences* 93 (2013). 3rd World Conference on Learning, Teaching and Educational Leadership, pp. 895–899.
- [KK15] **Kordyban, Ron & Kinash, Shelley. 2015.** Training Educators for This Brave New World. In: *Education Technology Solutions. For Principals / Educators / Network Administrators* 68 (2015): *Education Takes Flight: Using Drones to Engage Students*, pp. 52–55.
- [KKE] *Mikä on Koodikerho?* Internet-sivusto; Päivitetty: 2015. Saatavissa: [⟨http://koodikerho.fi/⟩](http://koodikerho.fi/), viitattu 20. joulukuuta 2015.

- [KKU] **Koodikoulu.** Internet-sivusto; Päivitetty: 2015. Saatavissa: <http://www.koodikoulu.fi/>, viitattu 20. joulukuuta 2015.
- [KOD] **Microsoft Research.** *Kodu Game Lab Community.* Internet Site. Available: <http://www.kodugamelab.com/>, visited on Mar. 6, 2016.
- [KOG] **Kogics.** *Kojo Home.* Internet Site. Available: <http://www.kogics.net/kojo>, visited on June 2, 2015.
- [KP05] **Kelleher, Caitlin & Pausch, Randy.** Lowering the Barriers to Programming. A Taxonomy of Programming Environments and Languages for Novice Programmers. In: *ACM Computing Surveys*. Vol. 37. 2. New York, New York, USA: ACM, 2005. Pp. 83–137.
- [LBR] **LibreLogo.org.** *Turtle Vector Graphics of LibreOffice.* Internet Site. Available: <http://librelogo.org/>, visited on June 2, 2015.
- [LEA13] **The LEAD Project. 2013.** *Super Scratch Programming Adventure! Learn to Program by Making Cool Games.* PDF eBook. San Francisco, California, USA: No Starch Press, Inc. 160 pp.
- [LEA16] **Leap Motion, Inc. 2016.** *Reach into new worlds.* Internet Site. Available: <https://www.leapmotion.com/>, visited on Apr. 12, 2016.
- [LEE13] **Lee, Cynthia Bailey.** Experience Report: CS1 in MATLAB for Non-majors, with Media Computation and Peer Instruction. In: *Proceeding of the 44th ACM Technical Symposium on Computer Science Education.* (Denver, Colorado, USA). SIGCSE '13. New York, New York, USA: ACM, 2013. Pp. 35–40.
- [LEG] **LEGO A/S. 2015.** *Home.* Internet Site. Available: <http://mindstorms.lego.com/>, visited on Nov. 23, 2015.
- [LEH15] **Lehtinen, Toni. 2015.** *Opettajat opettelevat nyt koodaamaan. Ensi syksynä ohjelmointi tulee alakouluihin.* Internet-sivu. Osasto: Kotimaa. Helsingin Sanomat, Sanoma Media Finland Oy. Päivitetty: lokakuu 2015. Saatavissa: <http://www.hs.fi/paivanlehti/26102015/a1445745902429>, viitattu 26. loka-kuuta 2015.
- [LGL] **Washington University in St. Louis.** *Looking Glass.* Internet Site. Available: <https://lookingglass.wustl.edu/>, visited on Mar. 1, 2016.
- [LH10] **Larkins, D. Brian & Harvey, William. 2010.** Introductory computational science using MATLAB and image processing. In: *Procedia Computer Science* 1.1 (2010). ICCS 2010, pp. 913–919.
- [LIG15] **LightBot, Inc. 2015.** *Lightbot.* Internet Site. Available: <https://lightbot.com/>, visited on Dec. 7, 2015.
- [LIU] **Liukas, Linda.** *Hello Ruby.* Internet Site. Available: <https://www.kickstarter.com/projects/lindaliukas/hello-ruby>, visited on June 24, 2015.
- [LLT14] **Lewis, Mark C. & Läufer, Konstantin & Thiruvathukal, George K.** Scala for Introductory CS and Parallelism (Abstract Only). In: *Proceedings of the 45th ACM Technical Symposium on Computer Science Education.* (Atlanta, Georgia, USA). SIGCSE '14. New York, New York, USA: ACM, 2014. P. 741.
- [LM14] **Liukas, Linda & Mykkänen, Juhani. 2014.** *KOODI2016. Ensiapua ohjelmoinnin opettamiseen peruskoulussa.* Pdf-kirja. Helsinki: Lönnberg Print. 144 s.
- [LMF] **Class LambdaMetafactory.** Internet Page. Oracle. Updated: 2015. Available: <https://docs.oracle.com/javase/8/docs/api/index.html?java/lang/invoker/LambdaMetafactory.html>, visited on Oct. 26, 2015.

- [LP16] **Linnenbrink-Garcia, Lisa & Patall, Erika A. 2016.** Motivation. In: *Handbook of Educational Psychology*. Ed. by Lyn Corno & Eric M. Anderman. 3rd ed. New York, New York, USA: Routledge, Taylor & Francis Group. Pp. 91–103.
- [LPD] **LilyPad Arduino.** Internet Site; Updated: 2016. Available: <http://lilypadarduino.org/>, visited on Feb. 7, 2016.
- [MA11] **Mayer, Richard E. & Alexander, Patricia A., eds. 2011.** *Handbook of Research on Learning and Instruction*. Educational Psychology Handbook Series. New York, New York, USA: Routledge, Taylor & Francis Group. 501 pp.
- [MAG+13] **Magerko, Brian & Freeman, Jason & McKlin, Tom & McCoid, Scott & Jenkins, Tom & Livingston, Elise.** Tackling Engagement in Computing with Computational Music Remixing. In: *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*. (Denver, Colorado, USA). SIGCSE '13. New York, New York, USA: ACM, 2013. Pp. 657–662.
- [MAL07] **Malpas, Phil. 2007.** *Capturing Colour*. Series: Basics Photography. Lausanne, Switzerland: AVA Publishing SA. 176 pp.
- [MAL14] **Malmi, Lauri. 2014.** Tools Research—What Is It? In: *ACM Inroads* 5.3 (2014), pp. 34–35.
- [MAM] **Shapes Robotics.** *Mama - an educational 3D programming language*. Internet Site. Available: <http://www.eytam.com/mama/>, visited on Mar. 6, 2016.
- [MAN93] **Mancoridis, Spiros.** A Multi-dimensional Taxonomy of Software Development Environments. In: *Proceedings of the 1993 Conference of the Centre for Advanced Studies on Collaborative Research: Software Engineering - Volume 1*. CASCOS '93. Toronto, Ontario, Canada: IBM Press, 1993. Pp. 581–594.
- [MAR14A] **Marji, Majed. 2014.** *Learn To Program with Scratch. A Visual Introduction to Programming with Games, Art, Science, and Math*. PDF eBook. San Francisco, California, USA: No Starch Press, Inc. 288 pp.
- [MAR14B] **Marshall, Gail. 2014.** New Halls of Knowledge. Modern learning environments accommodate time, space, place. In: *entrsekt. Where learning, technology and community meet* (2014), pp. 22–27.
- [MAS43] **Maslow, Abraham Harold. 1943.** A Theory of Human Motivation. In: *Psychological Review* 50.4 (1943), pp. 370–396.
- [MAT15] **MathWorks, Inc. 2015.** *MATLAB - The Language of Technical Computing*. Internet Site. Available: <http://www.mathworks.com/products/matlab/>, visited on Nov. 23, 2015.
- [MAY14A] **Mayer, Richard E. 2014.** Introduction to Multimedia Learning. In: *The Cambridge Handbook of Multimedia Learning*. Ed. by Richard E. Mayer. 2nd ed. New York, New York, USA: Cambridge University Press. Pp. 1–24.
- [MAY14B] **Mayer, Richard E., ed. 2014.** *The Cambridge Handbook of Multimedia Learning*. 2nd ed. New York, New York, USA: Cambridge University Press. 930 pp.
- [MBK15] **Manaris, Bill & Brown, Andrew R. & Kohn, Tobias.** Making Music with Computers: Creative Programming in Python (Abstract Only). In: *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*. (Kansas City, Missouri, USA). SIGCSE '15. New York, New York, USA: ACM, 2015. Pp. 705–705.
- [McC14] **McCue, Camille. 2014.** *Coding For Kids For Dummies*. Hoboken, New Jersey, USA: John Wiley & Sons, Inc. 384 pp.

- [McM15] **McMurray, Ian. 2015.** BYOD And Beyond. In: *Teaching&Learning UK. Technology for engaging minds* (2015): *Turn Teaching on Its Head*, pp. 10–12.
- [MCo] **Media Computation Teachers Website.** Internet Site; Updated: May 2016. Available: <http://www.mediacomputation.org/>, visited on Feb. 2, 2016.
- [MD14] **Morrison, Briana B. & DiSalvo, Betsy.** Khan Academy Gamifies Computer Science. In: *Proceedings of the 45th ACM Technical Symposium on Computer Science Education.* (Atlanta, Georgia, USA). SIGCSE '14. New York, New York, USA: ACM, 2014. Pp. 39–44.
- [MDv] **MonoDevelop.** Internet Site; Updated: 2016. Available: <http://www.monodevelop.com/>, visited on Mar. 14, 2016.
- [MES97] **Messmer, Hans-Peter. 1997.** *The Indispensable PC Hardware Book*. 3rd ed. 2nd printing, 1998. Harlow, England: Addison Wesley Longman Limited. 1384 pp.
- [MFO] **Rickitt Educational Media, Ltd.** *Magic Forest: A simple modelling environment for Early Years.* Internet Site. Available: <http://www.r-e-m.co.uk/logo/?comp=magicforest>, visited on Mar. 5, 2016.
- [MFR13] **Magana, Alejandra J. & Falk, Michael L. & Reese Jr., Michael J. 2013.** Introducing Discipline-Based Computing in Undergraduate Engineering Education. In: *Trans. Comput. Educ.* 13.4 (2013), 16:1–16:22.
- [Mic15A] **Microchip Technology, Inc. 2015.** *Microchip MCUs.* Internet Site. Available: <http://www.microchip.com/pic/>, visited on Nov. 23, 2015.
- [Mic15B] **Microsoft Corporation. 2015.** *Visual Studio - Microsoft Developer Tools.* Internet Site. Available: <https://www.visualstudio.com/>, visited on Nov. 23, 2015.
- [Mic15c] **Microsoft Corporation. 2015.** *Xbox / Official Site.* Internet Site. Available: <http://www.xbox.com/en-US/>, visited on Nov. 23, 2015.
- [Mic16] **Microsoft Corporation. 2016.** *Visual Studio Code - Code Editing. Redefined.* Internet Site. Available: <https://code.visualstudio.com/>, visited on Apr. 23, 2016.
- [MID15] **MIDI Manufacturers Association Incorporated. 2015.** *Tech Specs & Info.* Internet Site. Available: <http://www.midi.org/techspecs/index.php>, visited on Nov. 30, 2015.
- [MMG15] **Morrison, Briana B. & Margulieux, Lauren E. & Guzdial, Mark.** Subgoals, Context, and Worked Examples in Learning Computing Problem Solving. In: *Proceedings of the Eleventh Annual International Conference on International Computing Education Research.* (Omaha, Nebraska, USA). ICER '15. New York, New York, USA: ACM, 2015. Pp. 21–29.
- [MOR15] **Morgan, Nick. 2015.** *JavaScript for Kids. A Playful Introduction to Programming.* PDF eBook. San Francisco, California, USA: No Starch Press, Inc. 309 pp.
- [MOU15] **Moule, Reid. 2015.** Why Coding Matters. In: *Education Technology Solutions. For Principals / Educators / Network Administrators* 69 (2015): *The Power of Gaming in Education*, pp. 60–63.
- [MP06] **Meuser, Philipp & Pogade, Daniela. 2006.** *Construction and Design Manual. Wayfinding and Signage.* 2nd ed. Berlin, Germany: DOM publishers. 304 pp.

- [MR14] **Miller, Bradley N. & Ranum, David L. 2014.** *Python Programming in Context*. 2nd ed. Burlington, Massachusetts, USA: Jones & Bartlett Learning. 498 pp.
- [MUR+14] **Murray, Niall & Lee, Brian & Qiao, Yuansong & Muntean, Gabriel-Miro. 2014.** Multiple-Scent Enhanced Multimedia Synchronization. In: *ACM Trans. Multimedia Comput. Commun. Appl.* 11.1s (2014), 12:1–12:28.
- [MUS13] **Museovirasto. 2013.** *Missä on Esko, Suomen ensimmäinen tietokone?* Internet-sivu; Päivitetty: 2. tammikuuta 2013. Saatavissa: <http://www.kysymuseolta.fi/tekniikanmuseo/#!id=18>, viitattu 30. toukokuuta 2015.
- [MVN] **The Apache Software Foundation. 2016.** *Welcome to Apache Maven*. Internet Site. Available: <https://maven.apache.org/>, visited on May 8, 2016.
- [NAR+11] **Narumi, Takuji & Nishizaka, Shinya & Kajinami, Takashi & Tanikawa, Tomohiro & Hirose, Michitaka. 2011.** Augmented Reality Flavors: Gustatory Display Based on Edible Marker and Cross-modal Interaction. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. (Vancouver, British Columbia, Canada). CHI '11. New York, New York, USA: ACM, 2011. Pp. 93–102.
- [NBH] **Oracle Corporation. A Brief History of NetBeans.** Internet Page. Available: <https://netbeans.org/about/history.html>, visited on Apr. 22, 2016.
- [NEU15] **Neuron Fuel, Inc. 2015.** *Tynker / Programming courses fir kids*. Internet Site. Available: <http://www.tynker.com>, visited on Dec. 20, 2015.
- [NM11] **Nakamura, Hiromi & Miyashita, Homei. 2011.** Augmented Gustation Using Electricity. In: *Proceedings of the 2Nd Augmented Human International Conference*. (Tokyo, Japan). AH '11. New York, New York, USA: ACM, 2011. 34:1–34:2.
- [NTB] **Oracle Corporation. Welcome to NetBeans.** Internet Site. Available: <https://netbeans.org/>, visited on Nov. 23, 2015.
- [NXS] **The Nexus: A Community Project.** Internet Site. Available: <http://www.sonatype.org/nexus/>, visited on May 8, 2016.
- [OFF16] **Office of the President of the Republic of Finland. 2016.** *Presidentti Niinistö tutustui lasten koodikouluun*. Internet-sivu; Päivitetty: tammikuu 2016. Saatavissa: <http://www.presidentti.fi/Public/default.aspx?contentid=340470>, viitattu 7. helmikuuta 2015.
- [OGA] **OpenGameArt.Org.** Internet Site; Updated: May 12, 2007. Available: <http://opengameart.org/>, visited on May 16, 2016.
- [ORA16] **Oracle Corporation. 2016.** *JEP 222: jshell: The Java Shell (Read-Eval-Print Loop)*. Internet Page. Available: <http://openjdk.java.net/jeps/222>, visited on Mar. 16, 2016.
- [PAA13] **Paakki, Jukka. 2013.** *Opista tieteeksi – Suomen tietojenkäsittelytieteiden historia*. Pdf-kirja. Helsinki, Suomi: Tietojenkäsittelytieteen Seura ry. 250 s.
- [PAA15] **Paananen, Juha. 2015.** *Robogem — koko perheen hauska strategiapeli*. Internet-sivu. Saatavissa: <http://robogem.fi/>, viitattu 20. joulukuuta 2015.
- [PÄI15] **Päivänen, Pinja. 2015.** *Maailma olisi parempi paikka, jos naiset osaisivat koodata – näin väittävät Linda Liukas ja Tuuti Piippo*. Internet-sivu. Osasto: Raha. Helsingin Sanomat, Sanoma Media Finland Oy. Päivitetty: elokuu 2015. Saatavissa: <http://www.hs.fi/raha/a1440730162960>, viitattu 8. joulukuuta 2015.

- [PAJ08] **Paju, Petri. 2008.** *“Ilmarisen Suomi” ja sen tekijät. Matematiikkakomitea ja tietokoneen rakentaminen kansallisenä kysymyksenä 1950-luvulla.* Väitöskirja (monografia). Pdf-kirja. Historian laitos, Humanistinen tiedekunta, Turun yliopisto. 540 s.
- [PAN14] **Panoff, Robert Michael.** Computational Thinking for All: The Power and the Peril. In: *Proceedings of the 45th ACM Technical Symposium on Computer Science Education.* (Atlanta, Georgia, USA). SIGCSE '14. New York, New York, USA: ACM, 2014. Pp. 1–2.
- [PAR11] **Parlante, Nick. 2011.** *Nifty Image Puzzles.* Internet Page. Available: <http://nifty.stanford.edu/2011/parlante-image-puzzle/>, visited on Mar. 31, 2016.
- [PAR16] **Parallax, Inc. 2016.** *Scribbler 2 (S2) Robot - USB.* Internet Site. Available: <https://www.parallax.com/product/28136>, visited on Apr. 12, 2016.
- [PET00] **Petzold, Charles. 2000.** *CODE — Tietokonelaitteiden ja ohjelmien salainen maailma.* Jyväskylä, Finland: Gummerus Kirjapaino Oy. 670 pp.
- [PH03] **Peretz, Isabelle & Hyde, Krista L. 2003.** What is specific to music processing? Insights from congenital amusia. In: *Trends in Cognitive Sciences* 7.8 (2003), pp. 362–367.
- [PH14] **Parker, Jeanne E. & Hollister, Debra L. 2014.** The Cognitive Science Basis for Context. In: *Context in Computing. A Cross-Disciplinary Approach for Modeling the Real World.* Ed. by Patrick Brézillon & Avelino J. Gonzalez. New York, New York, USA: Springer Science+Business Media, LLC. Pp. 205–219.
- [PIN04] **Pink, Daniel H. 2004.** *Drive. The Surprising Truth About What Motivates Us.* Kindle eBook. New York, New York, USA: Riverhead Books, Penguin Group (USA), Inc.
- [PLA15] **The PLASA North American Technical Standards Office. 2015.** *Plasa Standards – Worldwide Standards for the Entertainment Industries. Published Documents.* Internet Site. Available: http://tsp.plasa.org/tsp/documents/published_docs.php, visited on Nov. 30, 2015.
- [PM09] **Pears, Arnold & Malmi, Lauri. 2009.** Values and Objectives in Computing Education Research. In: *ACM Transactions on Computing Education* 3 (2009), 15:1–15:6.
- [POR+09] **Pornpanomchai, Chomtip & Benjathanachat, Khanti & Prechaphuet, Suradej & Supapol, Jaruwat.** Ad-Smell: Advertising Movie with a Simple Olfactory Display. In: *Proceedings of the First International Conference on Internet Multimedia Computing and Service.* (Kunming, Yunnan, China). ICIMCS '09. New York, New York, USA: ACM, 2009. Pp. 113–118.
- [POR+13] **Porter, Leo & Guzdial, Mark & McDowell, Charlie & Simon, Beth. 2013.** Success in Introductory Programming: What Works? In: *Commun. ACM* 56.8 (2013), pp. 34–36.
- [PRÄ06] **Präkel, David. 2006.** *Composition.* Series: Basics Photography. Lausanne, Switzerland: AVA Publishing SA. 176 pp.
- [PRC] **Processing.org.** Internet Site; Updated: 2015. Available: <https://processing.org/>, visited on Nov. 23, 2015.
- [PRI14] **Pritchard, Alan. 2014.** *Ways of Learning. Learning Theories and Learning Styles in the Classroom.* 3rd ed. A David Fulton Book. Abingdon, Oxfordshire, UK: Routledge, Taylor & Francis Group. 143 pp.

- [PRI16] **Primo Toys. 2016.** *Hands-on coding without the screen for ages 3 and up.* Internet Site. Available: <http://www.primotoys.com/>, visited on Mar. 14, 2016.
- [PS13A] **Porter, Leo & Simon, Beth.** Retaining Nearly One-third More Majors with a Trio of Instructional Best Practices in CS1. In: *Proceeding of the 44th ACM Technical Symposium on Computer Science Education.* (Denver, Colorado, USA). SIGCSE '13. New York, New York, USA: ACM, 2013. Pp. 165–170.
- [PS13B] **Porter, Leo & Simon, Beth. 2013.** Fostering Creativity in CS1 by Hosting a Computer Science Art Show. In: *ACM Inroads 4.1* (2013), pp. 29–31.
- [PXY] **Hill, Katherine & Trower, Jake.** *Pixly: Programmatic Media Computation.* Internet Site. Available: <http://outreach.cs.ua.edu/pixly/>, visited on Mar. 6, 2016.
- [RAM+07] **Ramic, Belma & Chalmers, Alan & Hasic, Jasminka & Rizvic, Selma.** Selective Rendering in a Multi-modal Environment: Scent and Graphics. In: *Proceedings of the 23rd Spring Conference on Computer Graphics.* SCCG '07. New York, New York, USA: ACM, 2007. Pp. 147–151.
- [RAN+11] **Ranasinghe, Nimesha & Karunanayaka, Kasun & Cheok, Adrian David & Fernando, Owen Noel Newton & Nii, Hideaki & Gopalakrishnakone, Ponnampalam.** Digital Taste and Smell Communication. In: *Proceedings of the 6th International Conference on Body Area Networks.* (Beijing, China). BodyNets '11. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2011. Pp. 78–84.
- [RAN+13] **Ranasinghe, Nimesha & Cheok, Adrian & Nakatsu, Ryohei & Do, Ellen Yi-Luen.** Simulating the Sensation of Taste for Immersive Experiences. In: *Proceedings of the 2013 ACM International Workshop on Immersive Media Experiences.* (Barcelona, Spain). ImmersiveMe '13. New York, New York, USA: ACM, 2013. Pp. 29–34.
- [RAN+14A] **Ranasinghe, Nimesha & Lee, Kuan-Yi & Suthokumar, Gajan & Do, Ellen Yi-Luen.** Taste+: Digitally Enhancing Taste Sensations of Food and Beverages. In: *Proceedings of the 22Nd ACM International Conference on Multimedia.* (Orlando, Florida, USA). MM '14. New York, New York, USA: ACM, 2014. Pp. 737–738.
- [RAN+14B] **Ranasinghe, Nimesha & Lee, Kuan-Yi & Suthokumar, Gajan & Do, Ellen Yi-Luen.** The Sensation of Taste in the Future of Immersive Media. In: *Proceedings of the 2Nd ACM International Workshop on Immersive Media Experiences.* (Orlando, Florida, USA). ImmersiveMe '14. New York, New York, USA: ACM, 2014. Pp. 7–12.
- [RAN+14C] **Ranasinghe, Nimesha & Suthokumar, Gajan & Lee, Kuan Yi & Do, Ellen Yi-Luen.** Digital Flavor Interface. In: *Proceedings of the Adjunct Publication of the 27th Annual ACM Symposium on User Interface Software and Technology.* (Honolulu, Hawaii, USA). UIST'14 Adjunct. New York, New York, USA: ACM, 2014. Pp. 47–48.
- [RAN+15] **Ranasinghe, Nimesha & Suthokumar, Gajan & Lee, Kuan-Yi & Do, Ellen Yi-Luen.** Digital Flavor: Towards Digitally Simulating Virtual Flavors. In: *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction.* (Seattle, Washington, USA). ICMI '15. New York, New York, USA: ACM, 2015. Pp. 139–146.

- [RAS15] **Raspberry Pi Foundation. 2015.** *Raspberry Pi - Teach, Learn and Make with Raspberry Pi*. Internet Site. Available: [⟨https://www.raspberrypi.org/⟩](https://www.raspberrypi.org/), visited on Nov. 23, 2015.
- [RCD] **Robocode: Build the best - destroy the rest!** Internet Site; Updated: 2015. Available: [⟨http://robocode.sourceforge.net/⟩](http://robocode.sourceforge.net/), visited on Mar. 6, 2016.
- [RCK] **The Racket Language: A Programmable Programming Language.** Internet Site. Available: [⟨http://racket-lang.org/⟩](http://racket-lang.org/), visited on Nov. 18, 2015.
- [RDW13] **Rebelsky, Samuel A. & Davis, Janet & Weinman, Jerod.** Building Knowledge and Confidence with Mediascripting: A Successful Interdisciplinary Approach to CS1. In: *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*. (Denver, Colorado, USA). SIGCSE '13. New York, New York, USA: ACM, 2013. Pp. 483–488.
- [REN12] **Rendgen, Sandra. 2012.** *Information Graphics*. Köln, Germany: TASCHEN GmbH. 480 pp.
- [RES15] **Research Kitchen VOF. 2015.** *Learn to code with motivating interactive lessons*. Internet Site. Available: [⟨http://www.robomindacademy.com/⟩](http://www.robomindacademy.com/), visited on Mar. 6, 2016.
- [RIL12] **Riley, Derek.** Using Mobile Phone Programming to Teach Java and Advanced Programming to Computer Scientists. In: *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*. (Raleigh, North Carolina, USA). SIGCSE '12. New York, New York, USA: ACM, 2012. Pp. 541–546.
- [RLD13] **Ranasinghe, Nimesha & Lee, Kuan-Yi & Do, Ellen Yi-Luen.** FunRasa: An Interactive Drinking Platform. In: *Proceedings of the 8th International Conference on Tangible, Embedded and Embodied Interaction*. (Munich, Germany). TEI '14. New York, New York, USA: ACM, 2013. Pp. 133–136.
- [ROB+08] **Roberts, Eric & Bruce, Kim & Cutler, Robb & Cross, James & Grissom, Scott & Klee, Karl & Rodger, Susan & Trees, Fran & Utting, Ian & Yelling, Frank. 2008.** *ACM Java Task Force, Version 2.0*. Internet Site; Updated: Aug. 15, 2008. Available: [⟨http://cs.stanford.edu/people/eroberts/jtf/⟩](http://cs.stanford.edu/people/eroberts/jtf/), visited on Mar. 13, 2016.
- [ROB14] **Robot Turtles LLC. 2014.** *Robot Turtles | The Board Game that Teaches Programming to Kids. The Game for Little Programmers!* Internet Site. Available: [⟨http://www.robotturtles.com/⟩](http://www.robotturtles.com/), visited on Dec. 7, 2015.
- [RP08] **Rasala, Richard & Proulx, Viera. 2008.** *Java Power Tools*. Internet Site; Updated: Sept. 2, 2008. Available: [⟨http://www.ccs.neu.edu/jpt/⟩](http://www.ccs.neu.edu/jpt/), visited on Mar. 13, 2016.
- [RPG04] **Rich, Lauren & Perry, Heather & Guzdial, Mark.** A CS1 Course Designed to Address Interests of Women. In: *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*. (Norfolk, Virginia, USA). SIGCSE '04. New York, New York, USA: ACM, 2004. Pp. 190–194.
- [Rvw] **Robomatter, Inc. 2016.** *RobotVirtualWorlds*. Internet Site. Available: [⟨http://www.robotvirtualworlds.com/⟩](http://www.robotvirtualworlds.com/), visited on Mar. 6, 2016.
- [Rx] **ReactiveX. An API for Asynchronous Programming with Observable Streams.** Internet Site. Available: [⟨http://reactivex.io/⟩](http://reactivex.io/), visited on Oct. 26, 2015.
- [Ryz15] **Ryzac, Inc. 2015.** *Learn to code interactively, for free*. Internet Site. Available: [⟨http://www.codeacademy.com/⟩](http://www.codeacademy.com/), visited on Dec. 20, 2015.

- [SAM04] **Samara, Timothy. 2004.** *Typography Workbook. A Real-World Guide to Using Type in Graphic Design.* Beverly, Massachusetts, USA: Rockport Publishers, Inc. 240 pp.
- [SAM06] **Stone, Terry Lee & Adams, Sean & Morioka, Noreen. 2006.** *Color Design Workbook. A Real-World Guide to Using Color in Graphic Design.* Beverly, Massachusetts, USA: Rockport Publishers, Inc. 240 pp.
- [SAM07] **Samara, Timothy. 2007.** *Design Elements: A Graphic Style Manual.* Beverly, Massachusetts, USA: Rockport Publishers, Inc. 272 pp.
- [SAW06] **Sawyer, R. Keith, ed. 2006.** *The Cambridge Handbook of the Learning Sciences.* New York, New York, USA: Cambridge University Press. 626 pp.
- [SAX13] **Saxena, Shiti. 2013.** *Getting Started with SBT for Scala. Equip Yourself with a High-Productivity Work Environment Using SBT, a Build Tool for Scala.* PDF eBook. Series: Progressing. Birmingham, UK: Packt Publishing Ltd. 72 pp.
- [SBT] **Typesafe, Inc.** *SBT – The Interactive Build Tool.* Internet Site. Available: <http://www.scala-sbt.org/>, visited on Oct. 26, 2015.
- [Sc2120M3] **Scala 2.12.0-M3 Is Now Available!** Internet Page. Available: <http://www.scala-lang.org/news/2.12.0-M3>, visited on Oct. 26, 2015.
- [Sci16] **Scirra, Ltd. 2016.** *Construct 2.* Internet Site. Available: <https://www.scirra.com/construct2>, visited on Mar. 6, 2016.
- [SCk] **Nilsson, Richard. 2015.** *ScalaCheck: Property-Based Testing for Scala.* Internet Site. Available: <https://www.scalacheck.org/>, visited on Oct. 26, 2015.
- [SCR] **MIT Media Lab.** *Scratch.* Internet Site. Available: <https://scratch.mit.edu/>, visited on June 2, 2015.
- [ScW] **Scala Worksheet plugin for Eclipse.** Internet Site. Available: <https://github.com/scala-ide/scala-worksheet>, visited on Apr. 22, 2016.
- [SEP+15] **Seppälä, Otto & Ihanola, Petri & Isohanni, Essi & Sorva, Juha & Vihavainen, Arto.** Do We Know How Difficult the Rainfall Problem is? In: *Proceedings of the 15th Koli Calling Conference on Computing Education Research.* (Koli, Finland). Koli Calling '15. New York, New York, USA: ACM, 2015. Pp. 87–96.
- [SET97] **Sethi, Ravi. 1997.** *Programming Languages. Concepts & Constructs.* 2nd ed. Reading, Massachusetts, USA: Addison Wesley Longman, Inc. 325 (estimated).
- [SF15] **Suereth, Joshua & Farwell, Matthew. 2015.** *SBT in Action. The Simple Scala Build Tool.* PDF eBook. Shelter Island, New York, USA: Manning Publications Co. 261 pp.
- [SGP16A] **SGP Systems. 2016.** *SGP Baltie 4 C#.* Internet Site. Available: http://www.sgpsys.com/en/tour_b4.asp, visited on Mar. 6, 2016.
- [SGP16B] **SGP Systems. 2016.** *SGP Systems - Baltie C# 3D Game visual programming teaching tools for kids, children, youth and adults.* Internet Site. Available: <http://www.sgpsys.com/en/>, visited on Mar. 6, 2016.
- [SHE+14] **Sheard, Judy & Eckerdal, Anna & Kinnunen, Päivi & Malmi, Lauri & Nylén, Aletta & Thota, Neena.** MOOCs and Their Impact on Academics. In: *Proceedings of the 14th Koli Calling International Conference on Computing Education Research.* (Koli, Finland). Koli Calling '14. New York, New York, USA: ACM, 2014. Pp. 137–145.

- [SID15] **Scala IDE for Eclipse.** Internet Site; Updated: 2015. Available: <http://scala-ide.org/>, visited on Nov. 23, 2015.
- [SIM+10] **Simon, Beth & Kinnunen, Päivi & Porter, Leo & Zazkis, Dov.** Experience Report: CS1 for Majors with Media Computation. In: *Proceedings of the Fifteenth Annual Conference on Innovation and Technology in Computer Science Education*. (Bilkent, Ankara, Turkey). ITiCSE '10. New York, New York, USA: ACM, 2010. Pp. 214–218.
- [SIM15] **Simon. 2015.** *Emergence of Computing Education as a Research Discipline*. Article Dissertation. PDF eBook. Espoo, Finland: Department of Computer Science, School of Science, Aalto University. 220 pp.
- [SJR] **ScratchJr.** Internet Site. Available: <http://www.scratchjr.org/>, visited on Mar. 6, 2016.
- [SMI12] **Smiciklas, Mark. 2012.** *The Power of Infographics. Using pictures to communicate and connect with your audiences*. Indianapolis, Indiana, USA: Pearson Education, Inc. 199 pp.
- [SNP] **Snap!** Internet Site. Available: <https://snap.berkeley.edu/>, visited on Mar. 6, 2016.
- [SOL] **Solomon, Cynthia.** *Logo, Papert and Constructionist Learning*. Internet Page. Available: <https://logothings.wikispaces.com/>, visited on May 30, 2015.
- [SOR12] **Sorva, Juha. 2012.** *Visual Program Simulation in Introductory Programming Education*. Monograph. PDF eBook. Espoo, Finland: Department of Computer Science and Engineering, School of Science, Aalto University. 428 pp.
- [SQ] **Welcome To Squeak.** Internet Site. Available: <http://www.squeak.org/>, visited on June 2, 2015.
- [SRF] **Tudor, Vlad.** *SURF your LOGO code!* Internet Site. Available: <http://www.logointerpreter.com/>, visited on June 2, 2015.
- [SSA15] **Siegfried, Robert M. & Siegfried, Jason P. & Alexandro, Gina.** A Longitudinal Analysis of the Reid List of First Programming Languages. In: *2015 Proceedings of the Conference on Information Systems and Computing Education*. (Wilmington, North Carolina, USA). Information Systems & Computing Academic Professionals, 2015.
- [SSS14] **Smith, Neil & Sutcliffe, Clare & Sandvik, Linda.** Code Club: Bringing Programming to UK Primary Schools Through Scratch. In: *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*. (Atlanta, Georgia, USA). SIGCSE '14. New York, New York, USA: ACM, 2014. Pp. 517–522.
- [ST08] **Sloan, Robert H. & Troy, Patrick.** CS 0.5: A Better Approach to Introductory Computer Science for Majors. In: *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*. (Portland, Oregon, USA). SIGCSE '08. New York, New York, USA: ACM, 2008. Pp. 271–275.
- [STe] **ScalaTest. Simply Productive.** Internet Site. Artima, Inc. Updated: 2013. Available: <http://www.scalatest.org/>, visited on Oct. 26, 2015.
- [STe16] **Stencyl, LLC. 2016.** *Create Amazing Games Without Code*. Internet Site. Available: <http://www.stencyl.com/>, visited on Mar. 6, 2016.

- [STO+13] **Stonedahl, Forrest & Weintrop, David & Blikstein, Paulo & Shannon, Christine.** NetLogo: Teaching with Turtles and Crossing Curricular Boundaries (Abstract Only). In: *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*. (Denver, Colorado, USA). SIGCSE '13. New York, New York, USA: ACM, 2013. Pp. 763–763.
- [STS] **STEM to STEAM.** Internet Site. Rhode Island School of Design. Updated: 2015. Available: <http://stemtosteam.org/>, visited on Dec. 11, 2015.
- [SUG+10] **Sugimoto, Sayumi & Noguchi, Daisuke & Bannnai, Yuichi & Okada, Kenichi.** Ink Jet Olfactory Display Enabling Instantaneous Switches of Scents. In: *Proceedings of the 18th ACM International Conference on Multimedia*. (Firenze, Italy). MM '10. New York, New York, USA: ACM, 2010. Pp. 301–310.
- [SW08] **Sedgewick, Robert & Wayne, Kevin.** 2008. *Introduction to Programming in Java. An Interdisciplinary Approach*. Boston, Massachusetts, USA: Pearson Education, Inc. 723 pp.
- [SWD15] **Sedgewick, Robert & Wayne, Kevin & Dondero, Robert.** 2015. *Introduction to Programming in Java. An Interdisciplinary Approach*. Old Tappan, New Jersey, USA: Pearson Education, Inc. 792 pp.
- [TAC] **Turtle Academy. The Easy Way to Learn Programming.** Internet Site; Updated: 2015. Available: <https://turtleacademy.com/>, visited on June 2, 2015.
- [TAN99] **Tanenbaum, Andrew S.** 1999. *Structured Computer Organization*. 4th ed. Upper Saddle River, Jew Jersey, USA: Prentice Hall, Inc. 670 pp.
- [TEK15] **Tekniikan museo.** 2015. *Etusivu*. Internet-sivusto; Päivitetty: toukokuu 2015. Saatavissa: <http://www.tekniikanmuseo.fi/>, viitattu 30. toukokuuta 2015.
- [TEL14] **Tell, Stephanie.** 2014. Primary educators' free online computer science course. In: *Technology in Education. For Principals, Educators & Network Administrators 1* (2014): *3D Printing: Use Your Imagination*, p. 19.
- [TEW+08] **Tew, Allison Elliott & Dorn, Brian & Jr., William D. Leahy & Guzdial, Mark.** 2008. Context as Support for Learning Computer Organization. In: *J. Educ. Resour. Comput.* 8.3 (2008), 8:1–8:18.
- [TFG05] **Tew, Allison Elliott & Fowler, Charles & Guzdial, Mark.** Tracking an Innovation in Introductory CS Education from a Research University to a Two-year College. In: *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*. (St. Louis, Missouri, USA). SIGCSE '05. New York, New York, USA: ACM, 2005. Pp. 416–420.
- [TLK] **ToonTalk - Making programming child's play.** Internet Site. Available: <http://www.toontalk.com/>, visited on Mar. 6, 2016.
- [TMI15] **Tmi Koodirehtori / Juha Paananen.** 2015. *Koodikirja*. Internet-sivusto. Saatavissa: <http://www.koodikirja.fi/>, viitattu 20. joulukuuta 2015.
- [TNY] **Trower, Jake.** *Tunely: Program Sounds*. Internet Site. Available: <http://outreach.cs.ua.edu/tunely/>, visited on Apr. 9, 2016.
- [TON09] **Tondreau, Beth.** 2009. *Layout Essentials. 100 Design Principles for Using Grids*. Beverly, Massachusetts, USA: Rockport Publishers, Inc. 208 pp.
- [UTT+10] **Utting, Ian & Cooper, Stephen & Kölling, Michael & Maloney, John & Resnick, Mitchel.** 2010. Alice, Greenfoot, and Scratch – A Discussion. In: *Trans. Comput. Educ.* 10.4 (2010), 17:1–17:11.

- [VUK14] **Vukovic, Rebecca. 2014.** Carr driving flipped classroom technique to whole new level. In: *Technology in Education. For Principals, Educators & Network Administrators 1* (2014): *3D Printing: Use Your Imagination*, pp. 27–28.
- [WAT15A] **Waterloo Maple, Inc. 2015.** *Maple 2015*. Internet Site. Available: <http://www.maplesoft.com/products/maple/>, visited on Nov. 21, 2015.
- [WAT15B] **Waterworth, Chris. 2015.** Logic, Creativity and Problem Solving. In: *Teaching&Learning UK. Technology for engaging minds* (2015): *Cracking the Code*, pp. 22–24.
- [WAT15C] **Waterworth, Chris. 2015.** Pause, Rewind and Play. In: *Teaching&Learning UK. Technology for engaging minds* (2015): *Turn Teaching on Its Head*, pp. 18–20.
- [WOL15] **Wolfram Research, Inc. 2015.** *Wolfram Mathematica: Modern Technical Computing*. Internet Site. Available: <https://www.wolfram.com/mathematica/>, visited on Nov. 21, 2015.
- [WON15] **Wonder Workshop, Inc. 2015.** *Wonder Workshop | Home of Dash and Dot, robots that you can code*. Internet Page. Available: <https://www.makewonder.com/>, visited on Dec. 8, 2015.
- [WOR11] **World Wide Web Consortium. 2011.** *CSS Color Module Level 3*. Internet Page. W3C Recommendation 07 June 2011. Available: <https://www.w3.org/TR/css3-color/>, visited on Apr. 27, 2016.
- [YAC12] **Yacomuzzi, Paula. 2012.** *Logo Construction. How to Design and Build a Logo*. New York, New York, USA: HarperCollinsPublishers. 319 pp.
- [YAN+03] **Yanagida, Yasuyuki & Noma, Haruo & Tetsutani, Nobuji & Tomono, Akira.** An Unencumbering, Localized Olfactory Display. In: *CHI '03 Extended Abstracts on Human Factors in Computing Systems*. (Ft. Lauderdale, Florida, USA). CHI EA '03. New York, New York, USA: ACM, 2003. Pp. 988–989.
- [YG08] **Yarosh, Svetlana & Guzdial, Mark. 2008.** Narrating Data Structures: The Role of Context in CS2. In: *J. Educ. Resour. Comput.* 7.4 (2008), 6:1–6:20.
- [YoY16] **YoYo Games, Ltd. 2016.** *GameMaker: Studio*. Internet Site. Available: <http://www.yoyogames.com/gamemaker>, visited on Mar. 6, 2016.

Appendix A

Literature Data Providers

Table A.1 below presents the search engines and databases used to collect literature for this thesis.

Table A.1. The data providers utilized to obtain literature for this thesis.

Data Provider Service	(<i>Organization</i>)	⟨Internet Address⟩
Aaltodoc	<i>(Aalto University)</i>	⟨ https://aaltodoc.aalto.fi/ ⟩
ACM Digital Library	<i>(Association for Computer Machinery)</i>	⟨ http://dl.acm.org/ ⟩
Amazon		⟨ https://www.amazon.com/ ⟩
Amazon UK		⟨ https://www.amazon.co.uk/ ⟩
Doria	<i>(Several Finnish Universities)</i>	⟨ https://www.doria.fi/ ⟩
E-Thesis	<i>(University of Helsinki)</i>	⟨ https://ethesis.helsinki.fi/ ⟩
Google Scholar		⟨ https://scholar.google.fi/ ⟩
Google Search		⟨ https://www.google.fi/ ⟩
The Guardian		⟨ http://www.theguardian.com/ ⟩
Helsingin Sanomat		⟨ http://www.hs.fi/ ⟩
IEEE Explore	<i>(Institute of Electrical and Electronics Engineers)</i>	⟨ http://ieeexplore.ieee.org/ ⟩
Issuu		⟨ http://issuu.com/ ⟩
Jultika	<i>(University of Oulu)</i>	⟨ http://jultika.oulu.fi/ ⟩
Jyväskylä University Library		⟨ https://kirjasto.jyu.fi/collections/thesis-searches ⟩
ScienceDirect	<i>(Elsevier)</i>	⟨ http://www.sciencedirect.com/ ⟩
SpringerLink	<i>(Springer)</i>	⟨ http://link.springer.com/ ⟩
TamPub	<i>(University of Tampere)</i>	⟨ https://tampub.uta.fi/ ⟩
Taylor & Francis Online		⟨ http://www.tandfonline.com/ ⟩
Theseus	<i>(Finnish Universities of Applied Sciences)</i>	⟨ https://www.theseus.fi/ ⟩

Appendix B

Code Examples for IDEs Supporting Media Programming

This appendix presents simplistic unoptimized program code examples for some IDEs supporting the Media Programming approach. The related IDEs as well as the results of these examples are presented in § 3.5, and for the reader’s convenience, the corresponding program code is presented here. The order of presentation here follows that of § 3.5, and the source bitmap used in all pixel-based bitmap processing is presented in § 1.8.

The first example, Code Example B.1, is related to Jython Environment for Students. The function `compWiseOp()` processes a bitmap by applying the operations given as parameters to the corresponding red, green, and blue color components of every single pixel in the bitmap. The two “preset” filters, `nightVision()` and `orangeify()`, provide examples of calling the function. The three result bitmaps produced form the `versions` list, the content of which is then displayed by calling `openPictureTool()`. The resulting bitmaps are presented in Figure 3.1.

```
1  ORIG = 0; NEG = 1; MIN = 2; MAX = 3; MUL = 4

3  def compWiseOp(bitmapToProcess,
4                  redProcess, redParams,
5                  greenProcess, greenParams,
6                  blueProcess, blueParams):

8      RGB_MIN = 0; RGB_MAX = 255

10     mi = lambda p: RGB_MIN
11     ma = lambda p: RGB_MAX
12     id = lambda p: p["val"]
13     mu = lambda p: p["f"] * p["val"]
14     ne = lambda p: RGB_MAX - p["val"]

16     getOp = lambda op: {MIN: mi, MAX: ma, NEG: ne, MUL: mu}.get(op, id)
17     redOp = getOp(redProcess)
18     greenOp = getOp(greenProcess)
19     blueOp = getOp(blueProcess)

21     source = duplicatePicture(bitmapToProcess)
22     for p in source.getPixels():
23         redParams.update({"val": getRed(p)})
24         setRed(p, redOp(redParams))

26         greenParams.update({"val": getGreen(p)})
27         setGreen(p, greenOp(greenParams))
```

(continues on the next page...)

Code Example B.1. Trivial bitmap manipulation in Jython Environment for Students. The execution result of this code can be seen in Figure 3.1.

(continuing from the previous page)

```

28         blueParams.update({"val": getBlue(p)})
29         setBlue(p, blueOp(blueParams))

31     return source

34 def nightVision(bitmapToProcess):
35     return compWiseOp(original, MIN, {}, NEG, {}, MIN, {})

38 def orangefy(bitmapToProcess):
39     return compWiseOp(original, MUL, {"f": 2.5}, ORIG, {}, MUL, {"f": 0.2})

42 sourcePath = "0:\\Kehitys\\MastersThesis\\latex\\images\\proto\\imgproc\\sample4.jpg"
43 original = makePicture(sourcePath)

45 versions = [
46     nightVision(original),
47     orangefy(original),
48     compWiseOp(original, MAX, {}, NEG, {}, ORIG, {})
49 ]
50 for v in versions: openPictureTool(v)

```

Code Example B.1 (cont.). Trivial bitmap manipulation in Jython Enviroment for Students. The execution result of this code can be seen in Figure 3.1.

Code Example B.2 is for Kojo, and is actually one of its built-in examples with some numeric literals slightly adjusted. It uses turtle graphics to draw a simple recursive tree, in which each branch produces two shorter branches in differing angles with the constraint that all branches must be longer than one unit in length to be drawn (Figure 3.2).

```

1 def tree(distance: Double) {
2     if (distance > 1) {
3         setPenThickness(distance/7)
4         setPenColor(color(distance.toInt, math.abs(255-distance*3).toInt, 125))
5         forward(distance)
6         right(25)
7         tree(distance*1.0-3)
8         left(45)
9         tree(distance-10)
10        right(20)
11        back(distance)
12    }
13 }

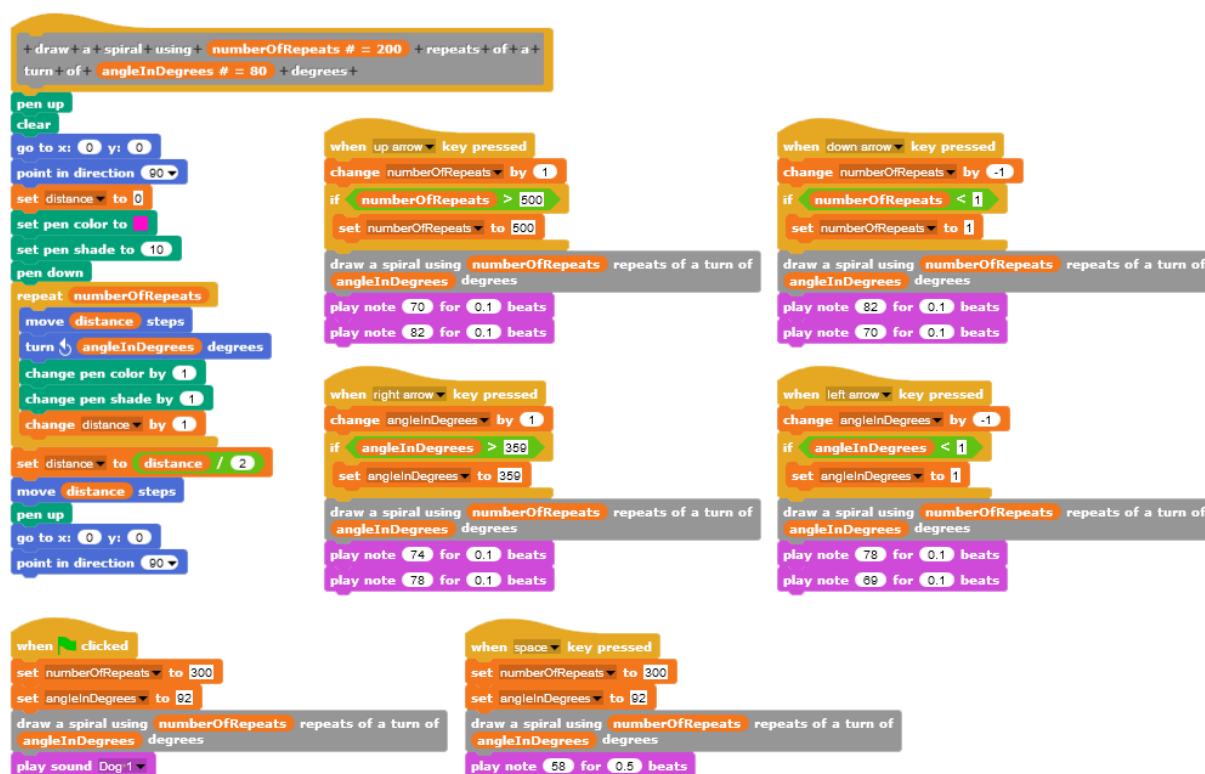
15 clear()
16 invisible()
17 setAnimationDelay(10)
18 penUp()
19 back(200)
20 penDown()
21 tree(90)

```

Code Example B.2. A simple recursive tree to be drawn as turtle graphics in Kojo. The execution result of this code can be seen in Figure 3.2.

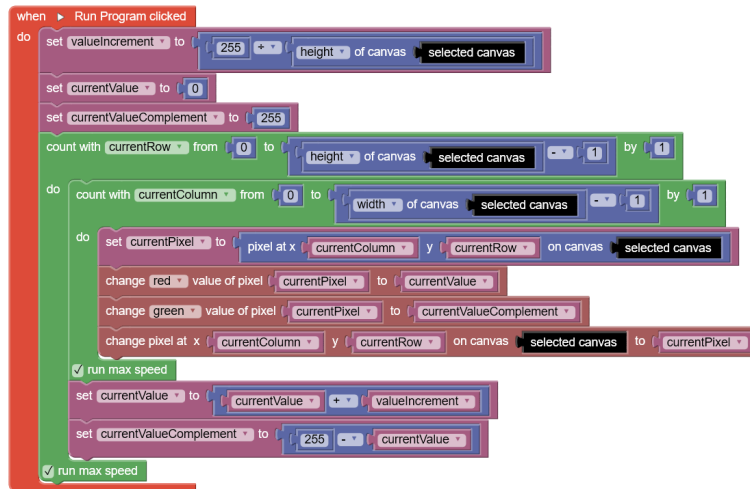
Code Example B.3 on the next page illustrates the usage of Snap! to draw simple turtle graphics and to play sounds. It has a self-defined block that implements a naive “forward-turn-repeat” spiral-drawing algorithm that is spiced up with continuous change of pen color and shade. The block is parameterized, so that both the number of lines drawn

and the angle to turn after each line can be given when calling the algorithm. The rest of the program is composed of six event handlers: Two for up and down arrow keys to increase or decrease number of lines to draw, respectively, two for right and left arrow keys to increase and decrease the amount of which to turn after each line drawn, one for the run button (the green flag) to initialize the program, and finally, one for space bar to reset it. All of these handlers use the custom-made block to redraw the result image, after which they play some sounds expressed in either notes or sound waveforms. Thus, the program can be used to explore how changing the two parameters affect to the resulting image (Figure 3.3). Both the dog image used as a turtle and the sound waveform of a dog barking are included in Snap!'s standard library.



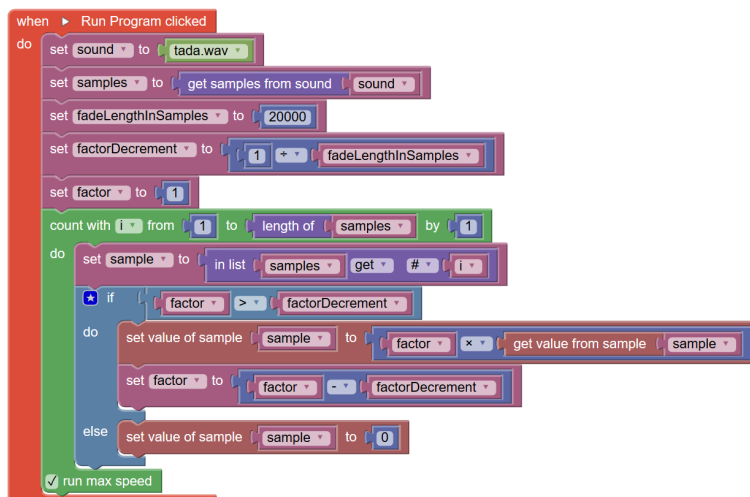
Code Example B.3. Turtle graphics and sound playing in Snap!. The image produced by this code can be seen in Figure 3.3.

In the spirit of the blockish appearance of the previous Snap! example, Code Example B.4 on the next page presents a naive algorithm to perform fades with color components using Pixly [PXY]. The code simply iterates through all pixels in the top-down fashion, and row-by-row increases the red color component from 0 to 255 and similarly decreases the green one from 255 to 0. Thus, the blue color component is the only one that is left with information from the original image. If the blue channel were zeroed, the resulting bitmap would be just a color gradient from green to red through yellow—none of the original image would be persisted. However, the lack of blue color in only some of the pixels in the processed bitmap brings out the silhouettes of the main subjects in the image as colors of the red–yellow–green gradient just described. The effect of this code can be seen in Figure 3.4.



Code Example B.4. Trivial bitmap manipulation in Pixly [PXY]. The execution result of this code can be seen in Figure 3.4.

For those who are interested in Pixly’s sibling called Tunely, Code Example B.5 below offers an example of performing a linear fade-out for a given sound waveform. The principle of the program code is similar to that of Code Example B.4 above: All sound samples participating into the fade-out are looped through while multiplying them by a factor that is linearly decremented from one to zero. Unfortunately, the effect of this code has to be tested in Tunely by the reader themselves—no sound examples are included in this thesis.



Code Example B.5. A trivial linear fade-out algorithm in Tunely [TNY]. To illustrate the if-else-block, the zeroing of the samples after the fade is not optimized into a separate loop that would follow the one performing the fade.

The next program code example is for EarSketch [FMV15]. The code just pseudorandomly selects some sound clips into several lists, places the contents of those lists onto the corresponding tracks by recursively looping over the lists, and finally sets the volumes of the tracks. An example of the resulting track content can be seen in Figure 3.5.

```

1  from earsketch import *

4  def placeSoundsOnTrack(soundlist, tracknum, start, cliplength):
5      if soundlist == []:
6          return

8      fitMedia(soundlist[0], tracknum, start, start+cliplength)
9      placeSoundsOnTrack(
10         soundlist[1:len(soundlist)],
11         tracknum, start+cliplength, cliplength)

14  init()
15  setTempo(124)

18  DRUMFOLDER = TMAINLOOP
19  BASSFOLDER = EABASS
20  SYNTHFOLDER = ELEAD
21  BLIPFOLDER = EIGHTTATARISFX

23  drumclips = []
24  bassclips = []
25  synthclips = []
26  blipclips = []

28  for i in range(4):
29      drumclips = drumclips + [ selectRandomFile(DRUMFOLDER) ]

31  for i in range(16):
32      bassclips = bassclips + [ selectRandomFile(BASSFOLDER) ]

34  for i in range(21):
35      synthclips = synthclips + [ selectRandomFile(SYNTHFOLDER) ]
36      blipclips = blipclips + [ selectRandomFile(BLIPFOLDER) ]

39  placeSoundsOnTrack(drumclips, 1, 1, 2)
40  placeSoundsOnTrack(bassclips, 2, 1, 0.5)
41  placeSoundsOnTrack(synthclips, 3, 1, 0.375)
42  placeSoundsOnTrack(blipclips, 4, 1.125, 0.375)

44  setEffect(1, VOLUME, GAIN, 0)
45  setEffect(2, VOLUME, GAIN, -6)
46  setEffect(3, VOLUME, GAIN, -12)
47  setEffect(4, VOLUME, GAIN, -9)

49  finish()

```

Code Example B.6. Clip-based sound waveform compositing in EarSketch [FMV15]. An example of execution results of this code can be seen in Figure 3.5.

The following two examples are for MathWorks MATLAB [MAT15]. Code Example B.7 is for “traditional” MATLAB without the Live Editor and is to be executed from the Command Window. All it does is create three versions of the original image—one with mutually swapped green and blue color channel, one with opposite gradients in red and green channels while blue channel is zeroed, and one black-and-white based on average of the three color channels. The result bitmaps can be seen in Figure 3.6a.

```

1  function[] = ImageProcessing1()
2      original = imread('0:\Kehitys\MastersThesis\latex\images\proto\imgproc\sample4.jpg');
3      v1 = original;
4      v1(:, :, 2) = original(:, :, 3);
5      v1(:, :, 3) = original(:, :, 2);

7      v2 = original;
8      v2(:, :, 3) = 0;
9      factor = 0.0;
10     factor_increment = 1.0 / size(v2, 2);
11     for x = 1:size(v2, 2)
12         v2(:, x, 1) = (1.0 - factor) * v2(:, x, 1);
13         v2(:, x, 2) = factor * v2(:, x, 2);
14         factor = factor + factor_increment;
15     end

17     v3 = original;
18     for x = 1:size(v3, 2)
19         for y = 1:size(v3, 1)
20             v3(y, x, :) = mean(v3(y, x, :));
21         end
22     end

24     figure('Name', 'Image Processing Examples', 'NumberTitle', 'off');

26     subplot(1, 4, 1);
27     imshow(original, []);
28     title('Original');

30     subplot(1, 4, 2);
31     imshow(v1, []);
32     title('Version 1');

34     subplot(1, 4, 3);
35     imshow(v2, []);
36     title('Version 2');

38     subplot(1, 4, 4);
39     imshow(v3, []);
40     title('Version 3');
41 end

```

Code Example B.7. Trivial bitmap manipulation with a traditional script in MathWorks MATLAB [MAT15]. The execution result of this code can be seen in Figure 3.6a.

Code Example B.8, in turn, is to be used in the Live Editor. There is one code cell for initialization and three others for producing bitmaps, of which one is based on constant color channel manipulations, one utilizes horizontal flipping and four color gradients, and one multiplies red and blue color channels by constant factors. The results of those code fragments can be seen in Figure 3.6b.

```

1  % Perform Initial Preparations
2  clear;
3  iptsetpref('ImshowInitialMagnification', 100.0);
4  path = '0:\Kehitys\MastersThesis\latex\images\proto\imgproc\sample4.jpg';
5  original = imresize(imread(path), 0.5);
6  small = imresize(original, 0.5);
7  display(sprintf('Path: %s\nOriginal size: %d x %d px; smaller size: %d x %d px.', ...
8               path, size(original, 1), size(original, 2), size(small, 1), size(small, 2)));

11 % Example 1
12 z = zeros(size(small, 1), size(small, 2), 1);
13 imshow([
14     original [
15         repmat(mean(small, 3, 'native'), [1 1 3]) ...
16         small(:, :, [1 3 2]));
17         small(:, :, [3 2 1]) ...
18         small(:, :, [2 1 3]) ...
19     ] [
20         cat(3, small(:, :, 1), z, z) ...
21         cat(3, repmat(mean(small(:, :, [1 2])), 3, 'native'), [1 1 2]), z);
22         cat(3, z, z, small(:, :, 3)) ...
23         cat(3, z, small(:, :, 2), z) ...
24     ] ...
25 ]);
26 clear z;

29 % Example 2
30 ybGrad = imresize(original, [0.9 * size(original, 1), 1.3 * size(original, 2)]);
31 rgGrad = cat(3, ybGrad(:, :, [1 2]), zeros(size(ybGrad, 1), size(ybGrad, 2), 1));
32 ybGrad = flipplr(ybGrad);
33 factor = 0.0;
34 factorIncrement = 1.0 / size(ybGrad, 2);
35 for x = 1:size(ybGrad, 2)
36     factorComplement = 1.0 - factor;
37     yellow = factor * ybGrad(:, x, 1);
38     ybGrad(:, x, [1 2 3]) = [yellow yellow (factorComplement * ybGrad(:, x, 3))];
39     rgGrad(:, x, [1 2]) = ...
40         [(factorComplement * rgGrad(:, x, 1)) (factor * rgGrad(:, x, 2))];
41     factor = factor + factorIncrement;
42 end
43 combined = [rgGrad ybGrad]; rotationAngle = 5;
44 rotated = imrotate(combined, rotationAngle, 'bilinear', 'loose');
45 rotationMask = ~imrotate(true(size(combined)), rotationAngle, 'bilinear', 'loose');
46 rotated(rotationMask & ~imclearborder(rotationMask)) = 255;
47 imshow(rotated);
48 clear ybGrad rgGrad factor factorIncrement x factorComplement ...
49     yellow combined rotationAngle rotated rotationMask resultImage;

52 % Example 3
53 result = repmat(mean(original, 3, 'native'), [1 1 3]);
54 result(:, :, 1) = min(255, 3.5 * result(:, :, 1));
55 result(:, :, 3) = min(255, 1.2 * result(:, :, 3));
56 imshow(result);
57 clear result;

```

Code Example B.8. Trivial bitmap manipulation with a live script in MathWorks MATLAB [MAT15]. The execution results of this code can be seen in Figure 3.6b.

Code Example B.9 demonstrates trivial bitmap manipulation in MathWorks MATLAB. It produces a single result bitmap which has several variations of the original image: Three small ones with color gradients, one average-based black-and-white version, and one negated version. The result bitmap can be seen in Figure 3.7.

```

1  restart
2  with(ImageTools):
3  with(ArrayTools):

5  hGrad := proc(imageToProcess :: Array, component :: integer) :: Array;
6      local img, i, j, factor;
7      img := Array(imageToProcess):
8      factor :=  $\frac{1}{\text{Height}(img)}$ :
9      for i to Height(img) do
10         for j to Width(img) do
11              $img_{i,j,component} := j \cdot factor$ :
12         end do;
13     end do;
14     return img
15 end proc:

17 blackAndWhiteAverage := proc(imageToProcess :: Array) :: Array;
18     local img, i, j;
19     img := Array(imageToProcess):
20     for i to Height(img) do
21         for j to Width(img) do
22              $img_{i,j,1..3} := \frac{img_{i,j,1} + img_{i,j,2} + img_{i,j,3}}{3}$ :
23         end do;
24     end do;
25     return img
26 end proc:

28 negativeImage := proc(imageToProcess :: Array) :: Array;
29     local img, i, j;
30     img := Array(imageToProcess):
31     for i to Height(img) do
32         for j to Width(img) do
33              $img_{i,j,1} := 1 - img_{i,j,1}$ :
34              $img_{i,j,2} := 1 - img_{i,j,2}$ :
35              $img_{i,j,3} := 1 - img_{i,j,3}$ :
36         end do;
37     end do;
38     return img
39 end proc:

41 original := Read("O:\Kehitys\MastersThesis\latex\images\proto\imageproc\sample4.jpg"):
42 small := Scale(original,  $1.. \frac{\text{Height}(original)}{2}$ ,  $1.. \frac{\text{Width}(original)}{2}$ ):
43 rowS1 := Concatenate(2, small, hGrad(small, 1)):
44 rowS2 := Concatenate(2, hGrad(small, 2), hGrad(small, 3)):
45 sq := Concatenate(1, rowS1, rowS2):
46 full := Concatenate(2, sq, blackAndWhiteAverage(original), negativeImage(original)):
47 Embed(full)

```

Code Example B.9. Trivial bitmap manipulation in Maple [WAT15A]. The execution result of this code can be seen in Figure 3.7.

The platform for Code Example B.10 is DrRacket. The code produces variants of the original bitmap by manipulating color channels and rotating, and finally combines them into one result bitmap that can be seen in Figure 3.8.

```

1  #lang racket
2  (require racket/os 2htdp/image math/statistics)

4  (define (map2colors img params fn)
5    (let* ([params (append params (list
6      (cons 'width (image-width img))
7      (cons 'height (image-height img)))]
8      [colors (map (curry fn params) (image->color-list img))])
9      (color-list->bitmap colors (image-width img) (image-height img))))

11 (define (get-alpha p) (cdr (assoc 'alpha p)))
12 (define (get-red-alpha p c)
13   (let ([o (cdr (assoc 'others p))])
14     (struct-copy color c [green o] [blue o] [alpha (get-alpha p)])))
15 (define (get-green-alpha p c)
16   (let ([o (cdr (assoc 'others p))])
17     (struct-copy color c [red o] [blue o] [alpha (get-alpha p)])))
18 (define (get-blue-alpha p c)
19   (let ([o (cdr (assoc 'others p))])
20     (struct-copy color c [red o] [green o] [alpha (get-alpha p)])))
21 (define (get-red-green p c)
22   (struct-copy color c [blue (cdr (assoc 'others p))] [alpha (get-alpha p)]))
23 (define (get-red-blue p c)
24   (struct-copy color c [green (cdr (assoc 'others p))] [alpha (get-alpha p)]))
25 (define (get-green-blue p c)
26   (struct-copy color c [red (cdr (assoc 'others p))] [alpha (get-alpha p)]))
27 (define (black-and-white-avg p c)
28   (let ([g (exact-truncate (mean (list
29     (color-red c) (color-green c) (color-blue c)))]))
30     (color g g g (get-alpha p))))

33 (define source-path (string-append
34   (if (string=? (gethostname) "BayOfBiscay") "D:\\Omat\\" "0:\\")
35   "Kehitys\\MastersThesis\\latex\\images\\proto\\imgproc\\sample4.jpg"))

37 (define original (scale 0.17 (bitmap/file source-path)))
38 (define original-large (scale 1.7 original))

40 (beside
41   (above
42     (apply beside (map (curry map2colors original '((others . 0) (alpha . 255)))
43       (list get-red-alpha get-green-alpha get-blue-alpha)))
44     (apply beside (map (curry map2colors original '((others . 0) (alpha . 255)))
45       (list get-red-green get-red-blue get-green-blue))))
46   (overlay/offset
47     (rotate 30 (flip-vertical (map2colors original-large
48       '((others . 255) (alpha . 255)) get-red-green)))
49     80 0
50     (rotate -30 (map2colors original-large
51       '((others . 255) (alpha . 255)) get-red-blue)))
52   (rotate 20 (overlay/offset
53     (flip-horizontal (map2colors original-large
54       '((alpha . 140)) black-and-white-avg))
55     10 -10
56     (flip-horizontal (map2colors original-large
57       '((others . 255) (alpha . 255)) get-green-blue)))))

```

Code Example B.10. Trivial bitmap manipulation in DrRacket [RCK]. The execution result of this code can be seen in Figure 3.8.

The last two code examples illustrate object embedding and bitmap processing in Wolfram Mathematica. The first one, Code Example B.11, produces the results visible in Figure 3.9a—that is, many different types of objects, most of which are editable straight in the code editor. Code Example B.12, in turn, integrates bitmaps with the program code and is reproduced here true to its presentation in Mathematica.

```

1 Clear["Global`*"];

3 temperatures = # → AirTemperatureData[
4   CountryData[#, "CapitalCity"], {2008, 8}, Max] & /@
5   DeleteCases[CountryData["Africa"], Entity["Country", "WesternSahara"]];


7 {Plot[Evaluate[Table[BesselJ[n, x], {n, 4}]], {x, 0, 10}, Filling → Axis],
8   CountryData["Finland", "Flag"],
9   ExampleData[{"TestImage3D", "CTengine"}],
10  ExampleData[{"Matrix", "GEMAT1"}],
11  ExampleData[{"Geometry3D", "Beethoven"}],
12  ExampleData[{"Sound", "CelloScale"}],
13  GeoRegionValuePlot[temperatures, PlotLegends → Histogram],
14  Plot3D[Sin[x + y ^ 2], {x, - 3, 3}, {y, - 2, 2}],
15  ExampleData[{"TestImage", "Airport"}],
16  CountryData["Finland", "PopulationGrowth"],
17  Row[Show[CountryData[#, "Shape"], ImageSize → {50, 50}] & /@ CountryData["G8"]],
18  CountryData["Finland", "Shape"], CountryData["G8"]}

```


Code Example B.11. A code example producing many kinds of multimedia that is integrated with code in Wolfram Mathematica [WOL15]. The execution result of this code can be seen in Figure 3.9a.





```





Clear["Global`*"];

aalto = ;









ImageAssemble[
  {aalto, Image[Map[ReplacePart[#, 1 -> 0.3 * #[[1]]] &, ImageData[aalto], {2}]],
    Image[Map[ReplacePart[#, 2 -> 0] &, ImageData[aalto], {2}]],
    Image[Map[ReplacePart[#, {2 -> #[[3]], 3 -> #[[2]]}] &, ImageData[aalto], {2}]]]}











sides = {3, 4, 5, 6};
colors = RGBColor /@
  {{0.26, 0.59, 0.82}, {0.96, 0.82, 0}, {0., 0.88, 0.02}, {0.89, 0.2, 0.5}}
{, , , }

twoD = {, , , };


twoDshadows = Blur[ImagePad[#, {{10, 10}, {10, 10}}, White], 4] & /@
  (ColorConvert[#, "Grayscale"] & /@ twoD);
{twoD, twoDshadows}

{{, , , }, {, , , }}













twoDpairs = Thread[{twoD, twoDshadows}]

{{, }, {, }, {, }, {, }}

ImageCollage[ImageCompose[#[[2]], #[[1]], {22, 28}] & /@ twoDpairs]



Graphics3D[{#[[1]], Opacity[0.5], PolyhedronData[#[[2]], "Faces"]},
  Boxed -> False, ImageSize -> {40, 40}] & /@ Tuples[{colors, PolyhedronData[9]}]

{, , , , ,
  , , , , , , }

```

Code Example B.12. Trivial bitmap manipulation in Wolfram Mathematica [WOL15] (as presented in Figure 3.9b).

Appendix C

Selected Resources by Topic

The following lists are given to aid the reader to find essential resources on topics related to this thesis. Some of these resources are also listed in the Bibliography—nevertheless, they are repeated here for the sake of completeness and for the reader’s convenience. Furthermore, the categories below are not mutually exclusive, so a single resource can be listed in several of them.

Eclipse Plug-In Development

Blewitt, Alex. 2013. *Eclipse 4 Plug-in Development by Example. How to develop, build, test, package, and release Eclipse plug-ins with features for Eclipse 3.x and Eclipse 4.x.* PDF eBook. Series: Beginner’s Guide. Birmingham, UK: Packt Publishing Ltd. 329 pp.

Blewitt, Alex. 2014. *Mastering Eclipse Plug-in Development. Build Modular Applications on Eclipse by Defining Custom Extension Points and Using OSGi Services.* PDF eBook. Series: Progressing. Birmingham, UK: Packt Publishing Ltd. 342 pp.

Kulkarni, Ram. 2013. *Instant Eclipse 4 RCP Development How-To.* PDF eBook. Series: Starting. Birmingham, UK: Packt Publishing Ltd. 55 pp.

Vogel, Lars. 2013. *Eclipse 4 RCP. The Complete Guide to Eclipse Application Development.* PDF eBook. 2nd ed. Vogella Series. Lars Vogel.

Education and Motivation

Auvinen, Tapio. 2015. *Educational Technologies for Supporting Self-Regulated Learning in Online Learning Environments.* Article Dissertation. PDF eBook. Espoo, Finland: Department of Computer Science and Engineering, School of Science, University. 200 pp.

Baillie, Caroline & Moore, Ivan, eds. 2004. *Effective Learning & Teaching in Engineering.* New York, New York, USA: Routledge, Taylor & Francis Group. 384 pp.

Bartlett, Steve & Burton, Diana. 2012. *Introduction to Education Studies.* 3rd ed. London, UK: SAGE Publications Ltd. 384 pp.

Bennedsen, Jens & Caspersen, Michael E. & Kölling, Michael, eds. 2008. *Reflections on the Teaching of Programming: Methods and Implementations.* PDF eBook. Vol. 4821. Berlin, Germany: Springer-Verlag GmbH. 260 pp.

Biggs, John & Tang, Catherine. 2011. *Teaching for Quality Learning at University.* 4th ed. Berkshire, UK: Open University Press, McGraw-Hill House. 389 pp.

- Carroll, John M., ed. 2014.** *Innovative Practices in Teaching Information Sciences and Technology. Experience Reports and Reflections.* PDF eBook. Cham, Switzerland: Springer International Publishing Switzerland. 238 pp.
- Corno, Lyn & Anderman, Eric M., eds. 2016.** *Handbook of Educational Psychology.* 3rd ed. New York, New York, USA: Routledge, Taylor & Francis Group. 481 pp.
- Fincher, Sally & Petre, Marian, eds. 2004.** *Computer Science Education Research.* Lisse, Netherlands: RoutledgeFalmer, Taylor & Francis The Netherlands. 239 pp.
- Fowler, Susan. 2014.** *Why Motivating People Doesn't Work . . . and What Does.* PDF eBook. San Francisco, California, USA: Berrett-Koehler Publishers, Inc. 218 pp.
- Fry, Heather & Ketteridge, Steve & Marshall, Stephanie, eds. 2015.** *A Handbook for Teaching & Learning in Higher Education. Enhancing Academic Practice.* 4th ed. Abingdon, Oxfordshire, UK: Routledge, Taylor & Francis Group. 452 pp.
- Hakulinen, Lasse. 2014.** *Gameful Approaches for Computer Science Education: From Gamification to Alternate Reality Games.* Article Dissertation. PDF eBook. Espoo, Finland: Department of Computer Science and Engineering, School of Science, Aalto University. 221 pp.
- Hazzan, Orit & Lapidot, Tami & Ragonis, Noa. 2014.** *Guide to Teaching Computer Science. An Activity-Based Approach.* PDF eBook. 2nd ed. London, UK: Springer-Verlag London Limited. 296 pp.
- Herala, Antti. 2015.** *Developing an Introductory Object-Oriented Programming Course.* Master's Thesis. PDF eBook. Lappeenranta, Finland: Degree Program in Computer Science, School of Business and Management, Lappeenranta University of Technology. 80 pp.
- Ihantola, Petri. 2011.** *Automated Assessment of Programming Assignments: Visual Feedback, Assignment Mobility, and Assessment of Students' Testing Skills.* Article Dissertation. PDF eBook. Espoo, Finland: Department of Computer Science and Engineering, School of Science, Aalto University. 228 pp.
- Illeris, Knud, ed. 2009.** *Contemporary Theories of Learning. Learning Theorists... in Their Own Words.* Abingdon, Oxfordshire, UK: Routledge, Taylor & Francis Group. 235 pp.
- Liukas, Linda & Mykkänen, Juhani. 2014.** *KOODI2016. Ensiapua ohjelmoinnin opettamiseen peruskoulussa.* Pdf-kirja. Helsinki: Lönnberg Print. 144 s.
- Mayer, Richard E., ed. 2014.** *The Cambridge Handbook of Multimedia Learning.* 2nd ed. New York, New York, USA: Cambridge University Press. 930 pp.
- Mayer, Richard E. & Alexander, Patricia A., eds. 2011.** *Handbook of Research on Learning and Instruction.* Educational Psychology Handbook Series. New York, New York, USA: Routledge, Taylor & Francis Group. 501 pp.
- Pink, Daniel H. 2004.** *Drive. The Surprising Truth About What Motivates Us.* Kindle eBook. New York, New York, USA: Riverhead Books, Penguin Group (USA), Inc.
- Pritchard, Alan. 2014.** *Ways of Learning. Learning Theories and Learning Styles in the Classroom.* 3rd ed. A David Fulton Book. Abingdon, Oxfordshire, UK: Routledge, Taylor & Francis Group. 143 pp.
- Sawyer, R. Keith, ed. 2006.** *The Cambridge Handbook of the Learning Sciences.* New York, New York, USA: Cambridge University Press. 626 pp.
- Seppälä, Otto. 2011.** *Advances in assessment of programming skills.* Article Dissertation. PDF eBook. Espoo, Finland: Department of Computer Science and Engineering, School of Science, Aalto University. 221 pp.
- Simon. 2015.** *Emergence of Computing Education as a Research Discipline.* Article Dissertation. PDF eBook. Espoo, Finland: Department of Computer Science, School of Science, Aalto University. 220 pp.

Sorva, Juha. 2012. *Visual Program Simulation in Introductory Programming Education*. Monograph. PDF eBook. Espoo, Finland: Department of Computer Science and Engineering, School of Science, Aalto University. 428 pp.

Torbert, Shane. 2012. *Applied Computer Science*. PDF eBook. New York, New York, USA: Springer Science+Business Media, LLC. 201 pp.

Waitley, Denis E. 2015. *The Psychology of Human Motivation*. Audible audio book. Wheeling, Illinois, USA: Nightingale-Conant.

Functional Programming

Backfield, Joshua. 2014. *Becoming Functional. Steps for Transforming Into a Functional Programmer*. PDF eBook. Sebastopol, California, USA: O'Reilly Media, Inc. 134 pp.

Bevilacqua-Linn, Michael. 2014. *Functional Programming Patterns in Scala and Clojure. Write Lean Programs for the JVM*. PDF eBook. P2.0—July 2014. Series: The Pragmatic Programmers. Dallas, Texas, USA: The Pragmatic Programmers, LLC. 236 pp.

Bjarnason, Rúnar Óli. 2015. *A Companion Booklet to Functional Programming in Scala. Chapter Notes, Errata, Hints, and Answers to Exercises*. PDF eBook. Boston, Massachusetts, USA: Runar LLC. 158 pp.

Blackheath, Stephen & Jones, Anthony. 2016. *Functional Reactive Programming (MEAP)*. PDF eBook. Manning Publications Co. 245 (estimated).

Chiusano, Paul & Bjarnason, Rúnar. 2014. *Functional Programming in Scala*. PDF eBook. Shelter Island, New York, USA: Manning Publications Co. 300 pp.

Khan, Aslam. 2016. *Grokking Functional Programming (MEAP)*. PDF eBook. Manning Publications Co. 475 (estimated).

Okasaki, Chris. 1998. *Purely Functional Data Structures*. Kindle eBook. Cambridge, UK: Cambridge University Press.

Image Processing

Burger, Wilhelm & Burge, Mark J. 2009. *Principles of Digital Image Processing. Core Algorithms*. PDF eBook. Series: Undergraduate Topics in Computer Science. London, UK: Springer-Verlag London Limited. 327 pp.

Burger, Wilhelm & Burge, Mark J. 2009. *Principles of Digital Image Processing. Fundamental Techniques*. PDF eBook. Series: Undergraduate Topics in Computer Science. London, UK: Springer-Verlag London Limited. 260 pp.

Efford, Nick. 2000. *Digital Image Processing: A practical introduction using Java*. Harlow, Essex, England: Jones & Bartlett Learning. 372 pp.

Moeslund, Thomas B. 2012. *Introduction to video and image processing. Building real systems and applications*. PDF eBook. Series: Undergraduate Topics in Computer Science. London, UK: Springer-Verlag London Limited. 227 pp.

Introductory Programming

Barnes, David J. & Kölling, Michael. 2005. *Objects First with Java. A Practical Introduction Using BlueJ*. 2nd ed. Essex, England, UK: Pearson Education Limited. 461 pp.

BBC. 2015. *Doctor Who: The Doctor and the Dalek teachers' pack*. Internet Page. Available: <http://www.bbc.co.uk/schools/0/computing/29614102>, visited on Dec. 7, 2015.

BBC. 2015. *The Doctor and the Dalek Game*. Internet Site. Available: <http://www.bbc.co.uk/programmes/articles/25S0qlFVD98csfFrvXcLhcr/the-doctor-and-the-dalek-game>, visited on Dec. 7, 2015.

Bloch, Stephen. 2013. *Picturing Programs. An Introduction to Computer Programming*. PDF eBook. 469 pp.

Breen, Derek. 2015. *Scratch For Kids For Dummies*. Hoboken, New Jersey, USA: John Wiley & Sons, Inc. 384 pp.

Briggs, Jason R. 2013. *Python for Kids. A Playful Introduction to Programming*. PDF eBook. San Francisco, California, USA: No Starch Press, Inc. 318 pp.

Brown, Andrew R. 2009. *Making Music with Java: An introduction to computer music, Java programming, and the jMusic library*. Lulu Press, Inc. 304 pp.

Code Monkey Games, LLC. 2014. *Code Monkey Island*. Internet Site. Available: <http://codemonkeyplanet.com/>, visited on Dec. 7, 2015.

CodeCombat Inc. 2015. *CodeCombat*. Internet Site. Available: <https://codecombat.com/>, visited on Dec. 20, 2015.

CodeMonkey Studios, Ltd. 2015. *Learn real coding*. Internet Site. Available: <https://www.playcodemonkey.com/>, visited on Dec. 20, 2015.

Dann, Wanda & Cooper, Stephen & Ericson, Barbara. 2010. *Exploring Wonderland. Java Programming Using Alice and Media Computation*. Upper Saddle River, New Jersey, USA: Pearson Education, Inc. 652 pp.

Dickins, Rosie. 2015. *Lift-the-Flap Computers and Coding*. Usborne Books. 16 pp.

Dickins, Rosie & Stowell, Louie & Melmoth, Jonathan. 2015. *Coding for Beginners using Scratch*. Usborne Books. 96 pp.

DK Publishing. 2014. *Computer Coding. An Introduction to Computer Programming*. Series: DK Workbooks. London, UK: Dorling Kindersley Limited. 40 pp.

DK Publishing. 2014. *Help Your Kids with Computer Coding. A Unique Step-by-Step Visual Guide, from Binary Code to Building Games*. London, UK: Dorling Kindersley Limited. 224 pp.

Felleisen, Matthias & Findler, Robert Bruce & Flatt, Matthew & Krishnamurthi, Shriram. 2015. *How to Design Programs*. Internet Site. 2nd Edition. Stable Release, Version 6.2.900.6. Updated: Aug. 6, 2015. Available: <http://www.ccs.neu.edu/home/matthias/HtDP2e/>, visited on Nov. 16, 2015.

Guzdial, Mark J. & Ericson, Barbara. 2005. *Introduction to Computing & Programming with Java. A Multimedia Approach*. Upper Saddle River, New Jersey, USA: Pearson Education, Inc. 558 pp.

Guzdial, Mark J. & Ericson, Barbara. 2011. *Problem Solving with Data Structures Using Java. A Multimedia Approach*. Upper Saddle River, New Jersey, USA: Pearson Education, Inc. 507 pp.

Guzdial, Mark J. & Ericson, Barbara. 2013. *Introduction to Computing and Programming in Python. A Multimedia Approach*. Kindle eBook. Third International. Essex, UK: Pearson Education Limited.

Hello Ruby Oy. 2015. *Home / Hello Ruby*. Internet Site. Available: <http://www.helloruby.com/>, visited on Dec. 7, 2015.

Her Interactive, Inc. 2016. *Nancy Drew: Codes & Clues*. Internet Site. Available: <http://www.herinteractive.com/nancy-drew-codes-clues/>, visited on Feb. 7, 2016.

Lewis, Mark C. 2012. *Introduction to the Art of Programming Using Scala*. Boca Raton, Florida, USA: CRC Press, Taylor & Francis Group. 936 pp.

LightBot, Inc. 2015. *Lightbot*. Internet Site. Available: <https://lightbot.com/>, visited on Dec. 7, 2015.

Manaris, Bill & Brown, Andrew R. 2014. *Making Music with Computers: Creative Programming in Python*. Boca Raton, Florida, USA: CRC Press, Taylor & Francis Group. 509 pp.

Marji, Majed. 2014. *Learn To Program with Scratch. A Visual Introduction to Programming with Games, Art, Science, and Math*. PDF eBook. San Francisco, California, USA: No Starch Press, Inc. 288 pp.

McCue, Camille. 2014. *Coding For Kids For Dummies*. Hoboken, New Jersey, USA: John Wiley & Sons, Inc. 384 pp.

Mikä on Koodikerho? Internet-sivusto; Päivitetty: 2015. Saatavissa: <http://koodikerho.fi/>, viitattu 20. joulukuuta 2015.

Miller, Bradley N. & Ranum, David L. 2014. *Python Programming in Context*. 2nd ed. Burlington, Massachusetts, USA: Jones & Bartlett Learning. 498 pp.

Morgan, Nick. 2015. *JavaScript for Kids. A Playful Introduction to Programming*. PDF eBook. San Francisco, California, USA: No Starch Press, Inc. 309 pp.

Payne, Bryson. 2015. *Teach Your Kids to Code. A Parent-Friendly Guide to Python Programming*. PDF eBook. San Francisco, California, USA: No Starch Press, Inc. 336 pp.

Richardson, Craig. 2015. *Learn to Program with Minecraft*. PDF eBook. San Francisco, California, USA: No Starch Press, Inc. 304 pp.

Robot Turtles LLC. 2014. *Robot Turtles / The Board Game that Teaches Programming to Kids. The Game for Little Programmers!* Internet Site. Available: <http://www.robotturtles.com/>, visited on Dec. 7, 2015.

Sedgewick, Robert & Wayne, Kevin. 2008. *Introduction to Programming in Java. An Interdisciplinary Approach*. Boston, Massachusetts, USA: Pearson Education, Inc. 723 pp.

Sedgewick, Robert & Wayne, Kevin & Dondero, Robert. 2015. *Introduction to Programming in Java. An Interdisciplinary Approach*. Old Tappan, New Jersey, USA: Pearson Education, Inc. 792 pp.

The LEAD Project. 2013. *Super Scratch Programming Adventure! Learn to Program by Making Cool Games*. PDF eBook. San Francisco, California, USA: No Starch Press, Inc. 160 pp.

Weinstein, Eric. 2015. *Ruby Wizardry. An Introduction to Programming for Kids*. PDF eBook. San Francisco, California, USA: No Starch Press, Inc. 327 pp.

Scala

Alexander, Alvin. 2013. *Scala Cookbook. Recipes for Object-Oriented and Functional Programming*. PDF eBook. Sebastopol, California, USA: O'Reilly Media, Inc. 694 pp.

Alexandre, Thomas. 2014. *Scala for Java Developers. Build Reactive, Scalable Applications and Integrate Java Code with the Power of Scala*. PDF eBook. Series: Progressing. Birmingham, UK: Packt Publishing Ltd. 262 pp.

Ansari, Saleem A. 2015. *Building a Recommendation Engine with Scala. Learn to Use Scala to Build a Recommendation Engine from Scratch and Empower Your Website Users*. Series: Starting. Birmingham, UK: Packt Publishing Ltd. 156 pp.

Backfield, Joshua. 2014. *Becoming Functional. Steps for Transforming Into a Functional Programmer*. PDF eBook. Sebastopol, California, USA: O'Reilly Media, Inc. 134 pp.

Bakker, Peter Hilton Erik & Canedo, Francisco. 2013. *Play For Scala. Covers Play 2*. Shelter Island, New York, USA: Manning Publications Co. 328 pp.

Bernhardt, Manuel. 2016. *Reactive Web Applications (MEAP). With Scala, Play, Akka, and Reactive Streams*. PDF eBook. Manning Publications Co. 325 (estimated).

Bevilacqua-Linn, Michael. 2014. *Functional Programming Patterns in Scala and Clojure. Write Lean Programs for the JVM*. PDF eBook. P2.0—July 2014. Series: The Pragmatic Programmers. Dallas, Texas, USA: The Pragmatic Programmers, LLC. 236 pp.

Bjarnason, Rúnar Óli. 2015. *A Companion Booklet to Functional Programming in Scala. Chapter Notes, Errata, Hints, and Answers to Exercises*. PDF eBook. Boston, Massachusetts, USA: Runar LLC. 158 pp.

Bugnion, Pascal. 2015. *Scala for Data Science. Leverage the Power of Scala with Different Tools to Build Scalable, Robust Data Science Applications*. Series: Learning. Birmingham, UK: Packt Publishing Ltd. 183 pp.

Burger, Wilhelm & Burge, Mark J. 2014. *A Beginner's Guide to Scala, Object Orientation and Functional Programming*. PDF eBook. Cham, Switzerland: Springer International Publishing Switzerland. 494 pp.

Chiusano, Paul & Bjarnason, Rúnar. 2014. *Functional Programming in Scala*. PDF eBook. Shelter Island, New York, USA: Manning Publications Co. 300 pp.

DeVore, Duncan K. & Walsh, Sean. 2015. *Reactive Application Development (MEAP)*. PDF eBook. Manning Publications Co. 400 (estimated).

Dirksen, Jos. 2015. *RESTful Web Services with Scala. Learn the Art of Creating Scalable RESTful Web Services with Scala*. Series: Learning. Birmingham, UK: Packt Publishing Ltd. 168 pp.

Ghosh, Debasish. 2011. *DSLs in Action*. PDF eBook. Stamford, Connecticut, USA: Manning Publications Co. 351 pp.

Hinojosa, Daniel. 2013. *Testing in Scala*. PDF eBook. Sebastopol, California, USA: O'Reilly Media, Inc. 148 pp.

Hunt, John. 2013. *Scala Design Patterns. Patterns for Practical Reuse and Design*. PDF eBook. Cham, Switzerland: Springer International Publishing Switzerland. 327 pp.

Jančauskas, Vytautas. 2016. *Scientific Computing with Scala. Learn to solve scientific computing problems using Scala and its numerical computing, data processing, concurrency, and plotting libraries*. Series: Learning. Birmingham, UK: Packt Publishing Ltd. 263 pp.

Khot, Atul. 2015. *Scala Functional Programming Patterns. Grok and Perform Effective Functional Programming in Scala*. Series: Mastering. Birmingham, UK: Packt Publishing Ltd. 275 pp.

- Kuhn, Roland & Allen, Jamie. 2015.** *Reactive Design Patterns (MEAP)*. PDF eBook. Manning Publications Co. 325 (estimated).
- Marivannan, Arun. 2015.** *Scala Data Analysis Cookbook. Navigate the World of Data Analysis, Visualization, and Machine Learning with over 100 Hands-On Scala Recipes*. PDF eBook. Series: Cookbook. Birmingham, UK: Packt Publishing Ltd. 234 pp.
- Nicolas, Patrick R. 2014.** *Scala for Machine Learning. Leverage Scala and Machine Learning to Construct and Study Systems That Can Learn from Data*. PDF eBook. Series: Learning. Birmingham, UK: Packt Publishing Ltd. 491 pp.
- Nikolov, Ivan. 2016.** *Scala Design Patterns. Grok with Scala to Write Efficient, Clean, Readable, Testable, and Extendible Code*. Series: Learning. Birmingham, UK: Packt Publishing Ltd. 332 pp.
- Nilsson, Rickard. 2014.** *ScalaCheck: The Definitive Guide. Property-Based Testing on the Java Platform*. Walnut Creek, California, USA: Artima, Inc. 148 pp.
- Odersky, Martin & Spoon, Lex & Venners, Bill. 2016.** *Programming in Scala. A comprehensive step-by-step guide*. PDF eBook. 3rd ed. Walnut Creek, California, USA: Artima, Inc. 859 pp.
- Petrella, Andy. 2013.** *Learning Play! Framework 2. Start Developing Awesome Web Applications with This Friendly, Practical Guide to the Play! Framework*. PDF eBook. Series: Learning. Birmingham, UK: Packt Publishing Ltd. 267 pp.
- Phillips, Andrew & Šerifović, Nermin. 2014.** *Scala Puzzlers. The Fun Path to Deeper Understanding*. PDF eBook. Walnut Creek, California, USA: Artima, Inc. 234 pp.
- Prokopec, Aleksandar. 2014.** *Learning Concurrent Programming in Scala. Learn the Art of Building Intricate, Modern, Scalable Concurrent Applications Using Scala*. PDF eBook. Series: Learning. Birmingham, UK: Packt Publishing Ltd. 339 pp.
- Raychaudhuri, Nilanjan. 2013.** *Scala in Action*. Shelter Island, New York, USA: Manning Publications Co. 416 pp.
- Roestenburg, Raymond & Bakker, Rob & Williams, Rob. 2015.** *Akka in Action (MEAP)*. PDF eBook. Manning Publications Co. 475 (estimated).
- Sathyanarayanan, Prasanna Kumar & Atreya, Suraj. 2016.** *Reactive Programming with Scala and Akka. Harness Reactive Programming to Build Scalable and Fault Tolerant Distributed Systems Using Scala and Akka*. Series: Learning. Birmingham, UK: Packt Publishing Ltd. 221 pp.
- Saxena, Shiti. 2013.** *Getting Started with SBT for Scala. Equip Yourself with a High-Productivity Work Environment Using SBT, a Build Tool for Scala*. PDF eBook. Series: Progressing. Birmingham, UK: Packt Publishing Ltd. 72 pp.
- Saxena, Shiti. 2015.** *Mastering Play Framework for Scala. Leverage the Awesome Features of Play Framework to Build Scalable, Resilient, and Responsive Applications*. PDF eBook. Series: Mastering. Birmingham, UK: Packt Publishing Ltd. 251 pp.
- Smart, John Ferguson. 2014.** *BDD in Action. Behavior-Driven Development for the Whole Software Lifecycle*. PDF eBook. Shelter Island, New York, USA: Manning Publications Co. 384 pp.
- Suereth, Joshua D. 2012.** *Scala in Depth*. Shelter Island, New York, USA: Manning Publications Co. 304 pp.
- Suereth, Joshua & Farwell, Matthew. 2015.** *SBT in Action. The Simple Scala Build Tool*. PDF eBook. Shelter Island, New York, USA: Manning Publications Co. 261 pp.

Theron, Vincent & Diamant, Michael. 2016. *Scala High Performance Programming. Write efficient, clean, and powerful Scala code and create high-performing applications that your users will love.* Series: Learning. Expected to be published in May 2016. Birmingham, UK: Packt Publishing Ltd. 187 pp.

Wampler, Dean & Payne, Alex. 2014. *Programming Scala. Scalability = Functional Programming + Objects.* PDF eBook. 2nd ed. Sebastopol, California, USA: O'Reilly Media, Inc. 548 pp.

Wyatt, Derek. 2013. *Akka Concurrency. Building Reliable Software in a Multi-Core World.* Walnut Creek, California, USA: Artima, Inc. 521 pp.