CODESTRATE PACKAGES:

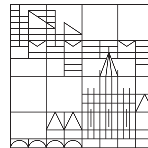
DESIGN AND EVALUATION OF A PACKAGE-BASED DEVELOPMENT ENVIRONMENT

Master Thesis

submitted by

Marcel Borowski

at the



Faculty of Sciences

Department of Computer and Information Science

1. Supervisor Prof. Dr. Harald Reiterer
2. Supervisor Prof. Dr. Clemens N. Klokmose

Advisor Johannes Zagermann

Konstanz, 2018

Marcel Borowski: *Codestrate Packages: Design and Evaluation of a Package-Based Development Environment* © 2018

ABSTRACT

Inspired by instrumental interaction and the concept of shareable dynamic media, the document-centric model describes a future where computation is (i) embedded within documents, (ii) shareable, malleable, and distributable among other documents and users, and (iii) independent from applications. Codestrate Packages implements this model by providing a package-based system to create user-extensible software. Its documents are shareable and collaboratively editable. It builds on Codestrates, a web-based computational notebook platform following the literate computing approach.

While it is a promising platform, Codestrates' or Codestrate Packages' influence on users and collaboration is still unexplored. To understand how collaboration unfolds in such a system, how computational notebooks affect programming, and how malleability and extensibility influence the development of applications, Codestrate Packages was deployed for 13 weeks during an introductory course on application development. During the course, pairs of students solved weekly programming assignments.

Data from weekly questionnaires, three focus groups consisting of students and teaching assistants, and keystroke-level log data were analyzed to facilitate the understanding of the subtleties of collaborative pair-based programming with computational notebooks. The findings reveal that there are distinct collaboration patterns. The preferred collaboration pattern varied between pairs and even varied within pairs throughout the 13 weeks. Further, the findings show that the linear structure of computational notebooks is beneficial for novices, however, can be restrictive for more experienced users. The reprogrammable nature and extensibility of the platform proved to be a double-edged sword as they, on the one hand, give users great expressive freedom, but on the other hand, bear the risk of users accidentally breaking the system.

Recognizing these benefits and barriers can help to guide the design of future computational notebooks.

PUBLICATIONS

Parts of this research were previously issued in the following term papers and publications:

Borowski, Marcel (2017). "Supporting Creative Group Work using Instrumental Interaction." Seminar paper to the Master-Project

Borowski, Marcel, Roman Rädle, and Clemens N. Klokmose (2018). "Codestrate Packages: An Alternative to "One-Size-Fits-All" Software." In: *CHI EA '18 Proceedings of the 2018 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. DOI: 10.1145/3170427.3188563

Borowski, Marcel (2018). "Codestrate Packages: An Alternative to "One-Size-Fits-All" Software." Master-Project Report

Parts of this research were submitted to be issued in the following publication:

Borowski, Marcel, Johannes Zagermann, Clemens N. Klokmose, Harald Reiterer, and Roman Rädle (2019). "Benefits and Barriers of Collaborative Computational Notebooks for Teaching Development of Interactive Systems." In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. CHI '19

// ACKNOWLEDGMENTS

I am grateful for all the support I received during my studies. Thanks to my advisor, Johannes Zagermann, for the countless discussions and invaluable feedback throughout the two-year venture of seminar, project, and thesis. Thanks to Roman Rädle for the feedback and ideas on my project and thesis, and for motivating me to push my limits. Thanks to my supervisors, Harald Reiterer and Clemens Klokmose, for making my stay abroad and, thus, the realization of Codestrate Packages possible.

Also, thanks to my friends for supporting me in various ways. Thanks to my sister for proof-reading my thesis, and for continuously improving my writing skills. Finally, a huge thanks to my parents for all their help, and for supporting me in all my decisions.

DECLARATION

I hereby declare that the attached master thesis on the topic

Codestrate Packages: Design and Evaluation of a Package-Based Development Environment

is the result of my own, independent work. I have not used any aids or sources other than those I have referenced in the document.

For contributions and quotations from the works of other people (whether distributed electronically or in hardcopy), I have identified each of them with a reference to the source or the secondary literature. Failure to do so constitutes plagiarism. I will also submit the master thesis electronically to the lecturer. Furthermore, I declare that the above-mentioned work has not been otherwise submitted as a master thesis.

Konstanz, 2018

Marcel Borowski

CONVENTIONS

Throughout this thesis the following conventions are used:

- The plural *we* will be used throughout this thesis instead of the singular *I*, even when referring to work that was primarily or solely done by the author.
- Unidentified third persons are always described in male form. This is only done for purposes of readability.
- Links to websites or homepages of mentioned products, applications or documents are shown in a footnote at the bottom of the corresponding page.
- References follow the Harvard citation format.
- The thesis is written in American English.

CONTENTS

1 INTRODUCTION 1
- 1.1 Motivation 2
- 1.2 Overview 3

2 THEORETICAL FOUNDATIONS 5
- 2.1 Human Activity Model 5
- 2.2 Reification, Polymorphism and Reuse 7
- 2.3 Instrumental Interaction 8
- 2.4 Ubiquitous Computing 10
- 2.5 Ubiquitous Instrumental Interaction 10
- 2.6 Literate Computing 11

3 RELATED WORK 13
- 3.1 Platforms 13
  - 3.1.1 Computational Notebooks 13
  - 3.1.2 Code Playgrounds 17
  - 3.1.3 Online Office Suites 18
- 3.2 Frameworks 20
  - 3.2.1 Webstrates 22
  - 3.2.2 Codestrates 24

4 SYSTEM AND SETUP 29
- 4.1 Codestrate Packages 29
  - 4.1.1 Packages 30
  - 4.1.2 Package Repositories 32
  - 4.1.3 Package Management 33
- 4.2 Interactive Systems Course 34
  - 4.2.1 Course Description 34
  - 4.2.2 Assignments 35
  - 4.2.3 Packages for Development 37
  - 4.2.4 Workflow 40

5 EVALUATION 45
- 5.1 Study 45
  - 5.1.1 Participants 45
  - 5.1.2 Procedure 47
  - 5.1.3 Apparatus 48
- 5.2 Data Analysis 48
  - 5.2.1 Questionnaires 50
  - 5.2.2 Interviews and Focus Groups 54
  - 5.2.3 Log Data 56
- 5.3 Findings 58
  - 5.3.1 Collaborative Working Styles 58

5.3.2 Web-based Computational Notebooks 63
5.3.3 Reprogrammability and Extensibility 66
5.4 Discussion 68
5.4.1 Collaborative Working Styles 68
5.4.2 Web-based Computational Notebooks 70
5.4.3 Reprogrammability and Extensibility 71

6 IMPLICATIONS AND FUTURE WORK 73
6.1 Implications for Design 73
6.1.1 General 73
6.1.2 Collaboration 75
6.1.3 Robustness 76
6.1.4 Package Management 77
6.2 Limitations and Future Work 78

7 CONCLUSION 81

A CONTENT OF THE FLASH DRIVE 83

B DEMOGRAPHIC QUESTIONNAIRE 85

C INTERVIEW AND FOCUS GROUP QUESTIONS 89
C.1 Weekly Interviews 89
C.2 Mid-Term Focus Group 89
C.3 End-Term Focus Group (Teaching Assistants) 90
C.4 End-Term Focus Group (Students) 91
C.5 End-Term Interview 92

D LOG DATA EXAMPLES 95
D.1 Package Management Log Data Example 95
D.2 Webstrates Log Data Example 97

E ASSIGNMENT ATTRIBUTES 99

BIBLIOGRAPHY 105

LIST OF FIGURES

Figure 1 Interaction between a user and a domain object 9
Figure 2 Weaving and tangling of a `WEB` file 12
Figure 3 Example of a computational notebook 14
Figure 4 User interface of Google Colaboratory 15
Figure 5 Computational notebook analysis by Rule et al. 16
Figure 6 Screenshot of the CodeCircle user interface . . 18
Figure 7 Examples of online office suites 19

Figure 8 Styles of collaborative writing 21
Figure 9 Synchronization of the DOM in Webstrates . . . 23
Figure 10 Schematic overview of the structure of a codestrate 27
Figure 11 Screenshot of a codestrate 28
Figure 12 Packages are added to documents 30
Figure 13 Three different section types 31
Figure 14 Different types of sharing packages 33
Figure 15 The structure of assignments 36
Figure 16 Execution of the code paragraph of an assignment 37
Figure 17 Screenshot of a computer science assignment . 38
Figure 18 The workflow of distributing and submitting assignments 43
Figure 19 Questions of the weekly questionnaire 52
Figure 20 Questions of the mid-term and end-term questionnaires 53
Figure 21 The generation process of sessions 58
Figure 22 Examples of collaborative working styles. Each row shows one assignment. Colored blocks in each row indicate sessions for S1, S2, and collaborative sessions. Each color indicates a different codestrate. 61
Figure 23 The template gallery in Microsoft Word 74
Figure 24 The track changes feature of ShareLaTeX . . . 76
Figure 25 A plugin page of WordPress 79

LIST OF TABLES

Table 1 Hierarchical structure of an activity 6
Table 2 List of packages recommended to the students 41
Table 3 Overview over the period of the course 47
Table 4 List of computer science assignments 49
Table 5 Overview over the data sources 50
Table 6 List of attributes used to assess working styles 59
Table 7 Primary working styles of pairs 62

LIST OF LISTINGS

Listing 1 General structure of a codestrate 26

LIST OF ACRONYMS

API Application Programming Interface

CSS Cascading Style Sheets

CSV Comma-Separated Values

DOM Document Object Model

FAQ Frequently Asked Questions

GIF Graphics Interchange Format

IDE Integrated Development Environment

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

JSON JavaScript Object Notation

PDF Portable Document Format

SVG Scalable Vector Graphics

URL Uniform Resource Locator

VCS Version Control System

WIMP Windows, Icons, Menus, Pointers

WYSIWYG What You See Is What You Get

1 INTRODUCTION

Of documents and applications

When solving a task with a computer, usually the first thing one does is to open an application. Then, within that application, a file is created or opened to be worked on, e.g., creating an image with a vector graphics program. When the scope of the task changes, for instance, if one needs to create a presentation with the newly created image, one often needs to switch the application, as the given task is outside the scope of the application's functionality. In order to use the image in the presentation program, however, it first needs to be exported, as the file is incompatible with the presentation program. If the task changes again, the juggling of applications continues even further. Hence, sharing files and working on them collaboratively is often not possible.

This file-based application-centric model has been a successful strategy for software development for a long time — and still is — as it eases the creation of encapsulated applications by making them independent from each other, meaning "that each application can be developed with the assumption that it exists in a vacuum" (Tchernavskij et al., 2017). While this reduces the complexity of applications, and thus the cost of creating them, it also causes users difficulties in handling them: working with a combination of multiple applications requires users to ex- and import content in order to move it between applications. An example of this would be situations in which "the technical tendencies of application-centric computing require non-standard knowledge workers to regularly abandon their preferred choice and switch to applications they are unfamiliar with or which change their ability to produce" (Nouwens and Klokmose, 2018). Moreover, applications often do not enable collaborative workflows.

Focus on the activity

Interaction models such as *instrumental interaction* and its extension *ubiquitous instrumental interaction* propose principles and concepts on how to address these problems (Beaudouin-Lafon, 2000; Klokmose and Beaudouin-Lafon, 2009). They introduce the idea of *instruments* — artifacts which act as mediators between users and *domain objects* with certain properties. These instruments can then, based on the activity, be used on a multitude of different domain objects without the need to re-implement them. The three design principles *reification*, *polymorphism*, and *reuse* support this notion even further by enabling users to fulfill a large set of tasks with as few instruments as possible (Beaudouin-Lafon and Mackay, 2000).

The document-centric model

Inspired by this interaction model, Codestrate Packages proposes a system to create extensible software using *packages*. The functional-

ity of packages — like that of instruments — can be reused in different scenarios. Content creation is transformed from an application-centric model into a document-centric model. Instead of moving documents from application to application to make use of the various functions of the different applications, the document itself contains the functionality. This functionality can be adjusted to the current needs by adding or removing packages. Further, documents as well as packages are shareable, malleable, and distributable, and make it possible for users to work in a collaborative manner.

As an extension of Codestrates (Rädle et al., 2017), Codestrate Packages also builds on the concept of literate computing (Pérez, 2013). Being an implementation of a computational notebook, Codestrates "blurs the distinction between the use and development of applications" (Rädle et al., 2017) and makes it possible to intertwine narrative text with executable code in the same document. Codestrates, which in turn builds on top of Webstrates (Klokmose et al., 2015) — a platform that implements the idea of *shareable dynamic media* —, makes Codestrate Packages inherently collaborative and accessible on a multitude of devices and operating systems, as it can be accessed via a web browser.

1.1 MOTIVATION

Influence on users is under-explored

While Codestrates has been evaluated according to properties proposed by Olsen (2007), it has not yet been evaluated by means of a user study. It is still unclear how the development environment of a computational notebook influences users while programming, where the benefits and barriers of literate computing lie, and how it compares to traditional file-based environments. Moreover, Codestrates is an inherently collaborative platform, which poses the question of how collaboration will unfold during the development of applications. Lastly, the malleability of Codestrates and Codestrate Packages allows users to alter the implementation of Codestrates itself as well as the provided packages, or to learn from them by examining their code. However, it is still unclear whether users would use these opportunities or would rather use the unaltered platform without tinkering with the implementation of the system.

This work aims to provide a deeper understanding of the preceding topics. It will explore these topics by investigating the following three research questions:

RQ1 How does collaborative programming in a computational notebook unfold?

RQ2 How does the structure of computational notebooks affect programming?

RQ3 How do the malleability and extensibility of Codestrate Packages influence programming?

Study "in the wild"

In order to answer these questions, Codestrate Packages was deployed for 13 weeks at the University of Konstanz during the introductory course *Interactive Systems*. The course focused on application development in human-computer interaction and the teaching of design patterns. The study allowed to observe how students and teaching assistants used the system and worked collaboratively on the development of interactive systems. As part of the course assignments, pairs of students were tasked to develop eight small interactive applications based on the textbook *Designing Interfaces* by Tidwell (2010).

1.2 OVERVIEW

The remainder of this thesis is split up into six chapters. After this introductory chapter, the chapter THEORETICAL FOUNDATIONS will give an introduction to theoretical models such as the human activity model, instrumental interaction and literate computing. The following chapter RELATED WORK will first present several related platforms to Codestrates as well as related research on collaborative web applications, and second, introduce the used frameworks Webstrates and Codestrates. The chapter SYSTEM AND SETUP will then give a detailed explanation of the platform Codestrate Packages, which was used for the conduction of the study, and present how it was further altered to be used within the course *Interactive Systems*. Subsequently, the chapter EVALUATION will illustrate how the study was conducted. It will list data sources, report on findings, and discuss the results in relation to the research questions. The chapter IMPLICATIONS AND FUTURE WORK will propose implications for the design of future systems and will suggest how future studies could resolve the limitations of this evaluation. Finally, the chapter CONCLUSION will summarize the thesis and give an outlook on future research.

2 THEORETICAL FOUNDATIONS

This chapter explains the theoretical foundations that are part of this thesis. The first two sections will focus on the *human activity model* and the three design principles *reification*, *polymorphism*, and *reuse*. Hereafter the following two sections will explain the interaction model *instrumental interaction* and its extension *ubiquitous instrumental interaction*. The latter extending the former using the vision of *ubiquitous computing* by Mark Weiser. The last section will conclude this chapter by explaining *literate programming* and *literate computing*.

2.1 HUMAN ACTIVITY MODEL

> *How can we understand why a bank teller has different needs for a user interface than those of casual users of a machine teller, or why a graphic designer needs a different user interface than a secretary?*
>
> — Bødker (1989)

Focus on the activity

This simple question is the starting point of the work by Bødker (1989). In her work, Bødker describes how the human activity model can be applied to user interfaces. She notes that during the design and composition of user interfaces the user interface itself is often the point of focus — not the task or the user.

However, by focusing on the user interface, the user interface also comes to the fore for users. They begin to interact more with the interface and less with the object that they originally meant to interact with. In her work *Through the Interface* Bødker explains "that a computer application, from the user's perspective, is not something that the user operates *on* but something that the user operates *through* on other objects or subjects" (Bødker, 1987). Users should thus focus on the task and object at hand, not the application or interface.

In order to get a better understanding of *activities*, the human activity model suggests to split activities into three levels (Bødker and Klokmose, 2011) (see Table 1):

ACTIVITY: At first, an activity addresses the question *Why?*. It is executed in order to achieve a personal goal or motive. An activity is usually composed of multiple *actions* which have to be executed in order to fulfill the activity. An example of an activity would be to write a letter to someone.

ACTION: Actions address the question *What?*. They are performed in order to fulfill subgoals of an overlying activity. The execution of actions happens consciously — a person actively thinks about the action and is aware of executing it. Actions, similarly to activities, are composed of multiple operations. For instance, writing a letter, putting it in an envelope, and finally sending it are three examples of actions.

OPERATION: Lastly, operations address the question *How?*. They are performed in order to fulfill individual actions. In contrast to activities and actions, operations are executed unconsciously. A person performs an operation in an automatic manner without actively thinking about it. Holding a pen while writing a letter is an example of an operation.

	QUESTION	GOAL	EXECUTION
ACTIVITY	Why?	Personal motive	Conscious
ACTION	What?	Fulfilling an activity	Conscious
OPERATION	How?	Fulfilling an action	Unconscious

Table 1: Hierarchical structure of an activity. (Bødker, 1989; Bødker and Klokmose, 2011)

Practice and conceptualization

Whether the execution of something is an action or an operation is not final and can change over time. Transferring actions into operations through practice is possible. When actions are performed multiple times, a person can imprint the sensomotoric procedures needed to perform the action. This process of learning allows people to perform an action "without thinking about it," i.e. to transform actions into operations. This transformation process can also take place the other way round: an operation can be performed while actively thinking about it. This means that the operation is performed as an action, which is called *conceptualization*. This can be done voluntarily, e.g., to optimize the operation, or involuntarily. The latter might happen when problems occur while one is carrying out the operation, for example, if one is trying to write with a pen which ink is empty. This is called a *breakdown*.

Mediation through artifacts

When executing operations a person can use artifacts in order to assist the task. A pen or scissors can, for example, act as artifacts. They act as mediators between a person and the object of interest. In the digital world, the user interface serves as such an artifact. For instance, the user interface of a text processing application can act as an artifact between the user and the text document. If problems occur while using such an artifact, the focus of the person shifts from the object of interest onto the artifact. This again causes a breakdown as the person's focus now lies on the artifact.

Bødker, therefore, recommends paying attention to the action of a user when designing user interfaces or artifacts in general. Understanding the activity as well as the underlying actions and operations is essential to create a user interface which fits the users' needs. That way breakdowns which hinder the user while performing a task can be prevented right away.

2.2 REIFICATION, POLYMORPHISM AND REUSE

Cluttered user interfaces

The ever-growing computational power of computers also caused the feature sets of applications to keep increasing over time. This development, however, does not only yield benefits: Graphical user interfaces become increasingly cluttered by the oversupply of functions causing users to spend more and more time dealing with the user interface than with their actual activity (Beaudouin-Lafon and Mackay, 2000).

Three design principles

Beaudouin-Lafon and Mackay present three design principles for the purpose of solving this problem. They allow to create simpler yet more powerful user interfaces:

REIFICATION: The first principle, *reification*, describes the process of reifying or objectifying concepts into objects. Through this objectification, concepts can become objects of interest and can thus be manipulated by users.

Beaudouin-Lafon and Mackay derive this principle from the interactions with everyday objects: When using a pencil, for instance, the pencil acts as an artifact that a person uses in order to write or draw on paper. However, when the pencil becomes dull and needs to be sharpened, the role of the pencil changes. The pencil becomes the object of interest, and the pencil sharpener becomes the new artifact altering the way in which the person interacts with the pencil.

Making it possible to change how artifacts behave — i.e. a sharp versus a dull pencil — allows to create hierarchies of objects and concepts. The same applies to computer commands that can be reified into objects, thus allowing users to have a smaller number of commands. These are more useful as they can be altered themselves by other commands.

POLYMORPHISM: *Polymorphism* describes the possibility to use the same command or artifact on different types of objects of interest. For example, commands such as *undo* and *redo* can both be used when editing text documents as well as when creating drawings in a graphics editor. This reduces the number of commands and makes the user interface more simple.

Besides using the same command on different types of objects, commands can also be used to modify groups of objects simultaneously.

This creates an increase in efficiency, e.g., when multiple objects need to be modified in the same way.

Like reification, polymorphism is also derived from the real world: a pencil, for example, can draw both on paper and on tables or walls. It can be used on different types of objects.

REUSE: The last principle *reuse* is divided into two different types of reuse: input reuse and output reuse.

The first type of reuse — input reuse — describes the reuse of user inputs. An example would be the command *redo*. The command allows users to reuse their input another time without having to enter it again. Input reuse is especially supported through polymorphism as polymorphism allows to use the same command on different types of objects without the need to modify the input.

Output reuse on the other side describes the reuse of the output or the resulting object. The command *copy & paste* is an example of this kind of reuse. It allows reusing a whole created object by duplicating it. In contrast to input reuse, output reuse is supported by the principle reification: When, for example, the formatting and style of a text in a word processor is reified into an object, this style object can be copied and applied to other texts without the need to manually create the style again.

The combination of principles is crucial

As the description of the principle reuse already suggests, the individual principles benefit from each other. This makes it clear, that a combination of the three principles is crucial to leverage the actual power of them.

2.3 INSTRUMENTAL INTERACTION

The interaction model *instrumental interaction* that was introduced by Beaudouin-Lafon (2000) in the year 2000 extends the principle of direct manipulation by Shneiderman (1983). In this model he makes use of concepts of the human activity model (cf. Section 2.1). The model focuses on classical WIMP (Windows, Icons, Menus, Pointers) user interfaces, which are used on single computers by single users.

Instruments and domain objects

As explained in the human activity model from Bødker (1989), one often interact with objects in the real world through artifacts that act as mediators. These artifacts are called *interaction instruments* — or short *instruments* — in the instrumental interaction model. The objects of interest, with which the users interacts, are called *domain objects*.

DOMAIN OBJECTS: The domain objects are the objects of interest of users; the objects they want to work with. In a graphics editor the image or graphic would be the domain object and in a word processor the text document. Domain objects thereby have attributes that de-

scribe them. These attributes range from simple numbers up to complex data types like textures. Complex attributes, like the mentioned textures, again can act as domain objects when the focus switches to them. This allows creating nested and hierarchical structures of domain objects.

When interacting with domain objects, either a single attribute or the whole object is modified. A single attribute could be modified when changing the color of an object. When deleting or duplicating an object, the whole object would be the target of the interaction.

INTERACTION INSTRUMENTS: Instruments are artifacts that users use as mediators between themselves and domain objects. Instruments consist of two parts. A physical part, the input device, and a logical part, the software.

The interaction of users with a domain objects using interaction instruments can be decomposed into multiple steps, as Figure 1 shows. The first step during an interaction is the *action* of users on an instrument. This action causes a direct reaction of the instrument that helps users to control their action. At the same time, the instrument transforms the action into a *command* that is modifying the domain object. The command causes the domain object to send a *response* after it was modified. This response finally is transformed by the instrument into visible *feedback* for the users.

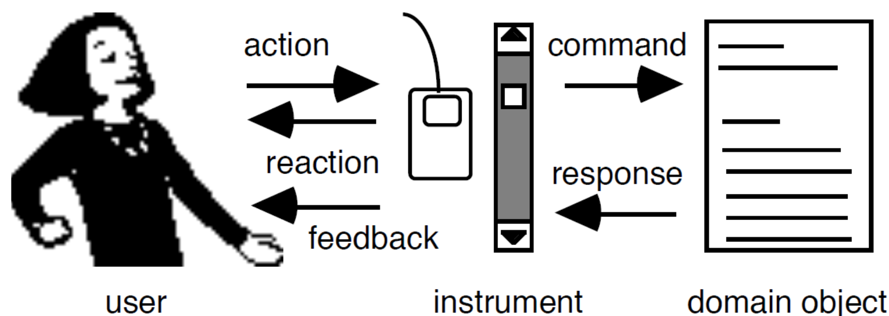


Figure 1: Interaction between a user and a domain object using interaction instruments. (Beaudouin-Lafon, 2000)

Activation of instruments

Before an instrument can be used by a user it first has to be activated. There are two ways how instruments can be activated: *spatial* activation and *temporal* activation. Instruments that are activated spatially need to be directly visible to the user. They can then be activated by simply clicking on them. An example of this would be a scrollbar, that is always visible in a window and can be used by navigating the mouse cursor on it and clicking. Temporal activation of an instrument, on the other hand, happens explicitly by activating the instrument. This means that while being inactive an instrument can be hidden from the user. For instance, the brush tool in a graphics

editor can be activated by clicking on a button in a toolbar and is hidden otherwise.

Reification of instruments

An essential property of instruments, however, is that they can be reified (cf. Section 2.2). For instruments, this can, for example, mean that multiple commands can be combined in a single instrument. Another possibility that is facilitated by reification is that instruments can become domain objects themselves. This allows so-called *meta-instruments* to modify the attributes of another instrument — similar to how a pencil can act as an instrument to write and as a domain object when being sharpened by a pencil sharpener. Meta-instruments, therefore, allow to change or adjust the function and behavior of instruments to a user's preference during runtime.

2.4 UBIQUITOUS COMPUTING

Direct perception of information

Almost 30 years ago in 1991, Weiser (1991) presented his vision of ubiquitous computing. In it he describes that "the most profound technologies are those that disappear" (Weiser, 1991) and that information should be perceived directly, not through technologies. He compares this idea with ordinary street signs: while driving one can perceive the information of a street sign without consciously thinking about that it is a street sign conveying the information. In the same way, he envisioned a future where computers would be used without actively perceiving them as computers.

Computers demand attention

One big problem that Weiser spotted in the use of technology and computers, was that they demanded a certain amount of attention in order to use them properly. As Bødker (1989) already described in her work, users often interact *on* the user interface and not *through* it (cf. Section 2.1). Instead, a user should be able to perceive the information a computer provides to them without having to think about the user interface or the computer itself.

Seamless usage

In his vision, Weiser eventually pictures a future in which computers are ubiquitous. They would be integrated seamlessly into rooms, be connected with each other wirelessly, and provide users with a seamless workflow that allows users to use whatever device they want to.

2.5 UBIQUITOUS INSTRUMENTAL INTERACTION

Distributed user interfaces

Building upon the interaction model instrumental interaction, the model *ubiquitous instrumental interaction* integrates the vision of ubiquitous computing into the original model. Its key addition to the model is the idea, that not only a single user but multiple users can interact with not one but multiple distributes user interfaces (Klokmose and Beaudouin-Lafon, 2009). Contrary to instrumental interaction which was designed for a single user and a single user interface,

ubiquitous instrumental interaction allows the distribution of interfaces onto multiple surfaces.

Ubiquitous instruments

By allowing multiple users and user interfaces, the requirements for instruments also needed to be extended. Instruments in ubiquitous instrumental interaction need to be easily interchangeable among multiple users and operate on all sorts of user interfaces. Klokmose and Beaudouin-Lafon again emphasize the importance of the polymorphism of instruments (cf. Section 2.2). Not only should it be possible to use instruments on different types of domain objects but also on different kinds of user interfaces. Even if the usage of an instrument on a specific domain object does not make sense in the first place, it is still important to allow these interactions in order to enable new idiosyncratic types of use of an instrument.

Challenges for both users and developers

These extensions to the interaction model, however, also bring new challenges. On the one hand, users will face challenges comprehending the model. Being able to use the same instrument among different types of objects and even different types of user interfaces is not common in everyday computer usage. Usually, every application provides their own tools that can only be used within that very application. On the other hand, developers will be confronted with both hardware and software challenges when implementing these concepts. Reification and polymorphism allow for a sheer endless number of combinations of instruments, domain objects, and user interfaces. It will be a tedious task to implement this variety of possibilities in a consistent way so that users can properly use them.

2.6 literate computing

Works of literature

Computer programs and algorithms are often written with the computer in mind — not other human beings. As computers became more and more powerful and computer programs became larger and larger, also a better way of documentation was needed. This lead to the programming paradigm *literate programming* of Knuth (1984). In it, he describes the idea to consider programs as works of literature. Instead of having a separate documentation of program code he proposes the WEB system. It combines both the programming language Pascal and the typesetting system TeX of his in the new `WEB` format. In it TeX code can be written alongside Pascal code. The WEB system then would use the `WEB` file to both *weave* a `TEX` and *tangle* a `PAS` file out of it (see Figure 2).

Exploratory computing

The target group of the WEB system and literate programming are system programmers (Knuth, 1984). Thus, before the execution of the `WEB` code it always needed to be tangled into Pascal code. Scientists who investigate data in an exploratory way, however, often need to execute small fragments of code in an iterative manner (Pérez, 2013). Therefore, based on literate programming, the paradigm *literate com-*

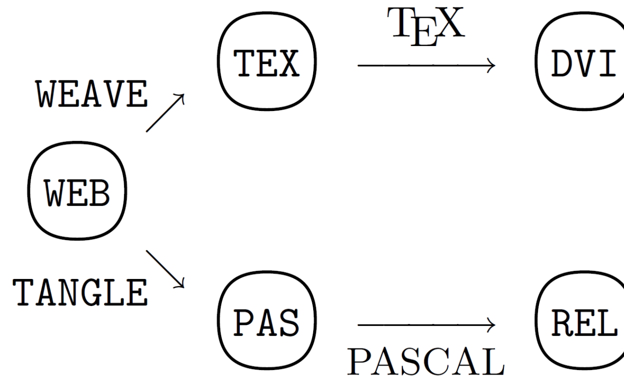


Figure 2: Weaving and tangling of a WEB file in order to generate a TEX and a PAS file. (Knuth, 1984)

puting was introduced. Pérez (2013) defines it as "the weaving of a narrative directly into a live computation, interleaving text with code and results to construct a complete piece that relies equally on the textual explanations and the computational components." An implementation of this paradigm can be found in computational notebooks such as Jupyter Notebook (Kluyver et al., 2016), which will be discussed in more detail in Section 3.1.1.

Narrowing the gap

Besides the motivation to provide a system or tool for a better documentation and explanation of programs, there are also other motives and potentials of such a paradigm: In their work Codestrates, Rädle et al. (2017) describe how such systems can also be used to narrow "the gap between developing and using applications" and thereby go "beyond the paradigmatic application-document model." Usually when one develops an application one is provided with a development environment like an IDE (Integrated Development Environment) or code editor for the development of the application. In order to use the application, one then has to either compile the code to a binary file that can be executed or open the file in another application — say opening an HTML (Hypertext Markup Language) file in a web browser. This shows that there is a clear separation of developing and using an application. This gap can be narrowed by providing the possibility to execute code directly within an environment and use the results of that execution.

3 RELATED WORK

This chapter will now address work related to this thesis. The first section Platforms will start with various related platforms to Codestrates and Codestrate Packages. These platforms all have different points of commonality with Codestrates. They are, for example, collaborative, use web technologies or follow similar programming paradigms—some of them even share more than one property. Afterwards, the section Frameworks will cover the platforms Webstrates and Codestrates in more detail—both regarding their underlying concepts and technical implementation. They are the basis of Codestrate Packages which will be explained in the next chapter.

3.1 PLATFORMS

Codestrates and its extension Codestrate Packages offer a platform that is equipped with a customizable range of functionality that stores less unused functionality, does not silo functionality in an application but supports shareable and malleable functionality that is part of the document. This section will look into platforms with similar characteristics and elements. It will first give an overview of computational notebooks with regard to the existing platforms and the research that has been done on them; next it will introduce code playgrounds and their features for fast prototyping. Lastly, it will cover online office suites: they are another type of platform that allows users to write texts collaboratively using web technologies.

3.1.1 *Computational Notebooks*

Computational notebooks are an implementation of the literate computing paradigm (cf. Section 2.6). They allow for the weaving of code and narrative within the same document.

Jupyter Notebook

Jupyter Notebook[1] by Kluyver et al. (2016) is an implementation of computational notebooks which supports the literate computing paradigm. It allows users to create cells for narrative text, code, and visualizations (see Figure 3). Text cells allow for rich-text editing and formatting using the markup language Markdown. Code cells run Python code and can output plain text or visualizations underneath the respective cell. Jupyter Notebook also allows converting notebooks into HTML, LaTeX or PDF (Portable Document Format) files.

1 Jupyter Notebook: `http://jupyter.org/` (accessed November 15, 2018)

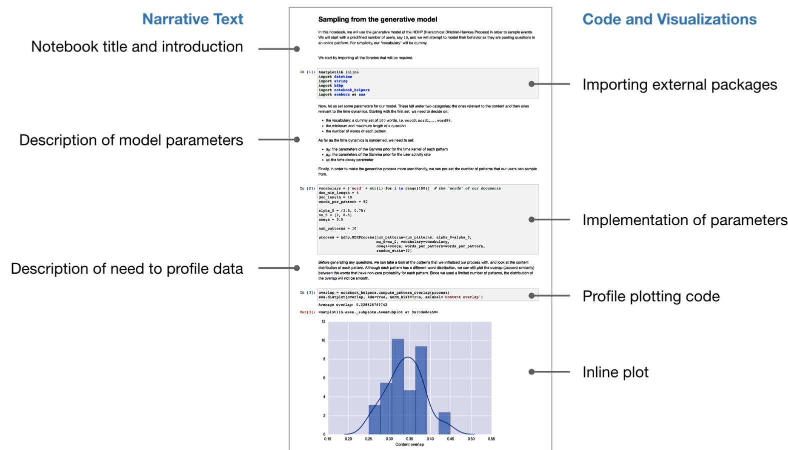


Figure 3: Example of a computational notebook. The displayed notebook features various cells for narrative text, code, and visualizations. (Rule et al., 2018)

Observable

Observable[2] is another implementation of computational notebooks. It omits the separation of text and code cells and allows users to create *reactive* cells. Reactive cells can contain text in various formats such as HTML, Markdown, or executable code. In contrast to Jupyter where cells can be executed manually in any order, the code execution of Observable is reactive, which means when editing code in a cell all dependent cells are automatically updated. Similar to Jupyter, the front-end of Observable is accessible in a web browser. While Jupyter executes code on a server, Observable runs computations within the browser using JavaScript as its programming language.

Google Colaboratory

Google Colaboratory[3] is another platform that is accessible in a web browser (see Figure 4). Building on top of Jupyter Notebook, notebooks in Colaboratory are composed of text and code cells. In contrast to Observable, users can use Python as a programming language in Colaboratory. The code execution, however, happens in the cloud, so users do not need to set up any application on their devices. Furthermore, Google Colaboratory allows for real-time collaboration similar to Google Docs[4]. Google Colaboratory stores notebooks within Google Drive[5], and just like other web content, these notebooks can be embedded into websites. Platforms like Distill[6] serve as an excellent example of how this embedding can be used to create interactive publications that can be accessed in a web browser.

Idyll

Idyll by Conlen and Heer (2018) dedicates itself to journalists who want to publish interactive articles. It allows them to create interactive narratives with a markup language and JavaScript components. This

2 Observable: https://beta.observablehq.com/ (accessed November 15, 2018)
3 Google Colaboratory: https://colab.research.google.com/ (accessed November 15, 2018)
4 Google Docs: https://docs.google.com/ (accessed November 15, 2018)
5 Google Drive: https://www.google.com/drive/ (accessed November 15, 2018)
6 Distill: https://distill.pub/ (accessed November 15, 2018)

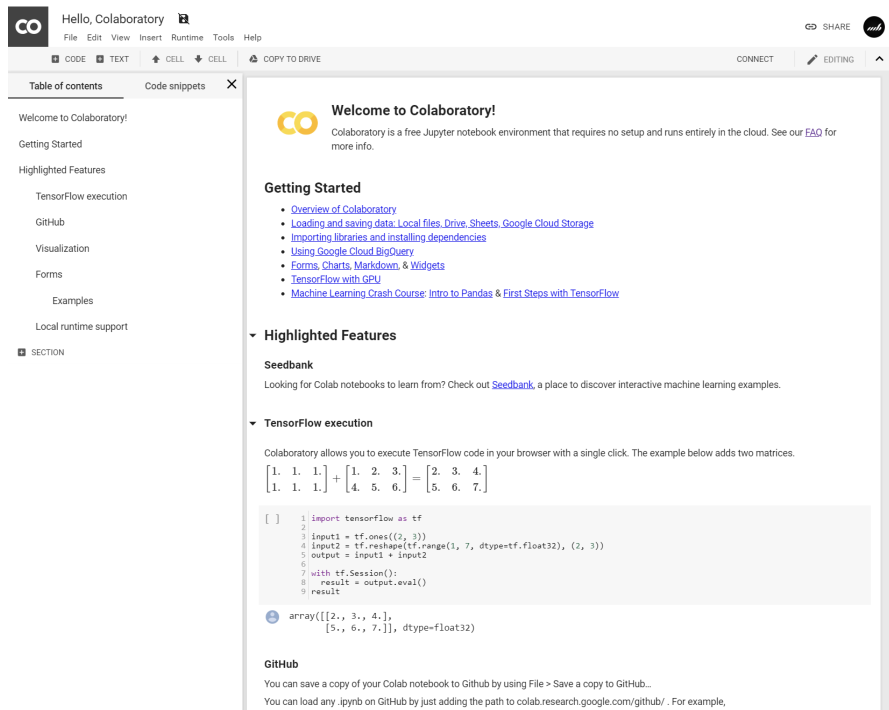


Figure 4: User interface of Google Colaboratory.

separation of narrative and executable code is similar to the text cell and code cell structure of Jupyter Notebook and Google Colaboratory.

Codestrates

Codestrates is a web-based platform that enables users to develop applications or create documents with embedded computation resembling a notebook-like environment (Rädle et al., 2017). These documents are called *codestrates*. A codestrate is structured in sections and paragraphs. Codestrates offers four types of paragraphs: body paragraphs, code paragraphs, style paragraphs, and data paragraphs. Similar to Google Colaboratory, Codestrates allows for real-time collaboration. This is the case not only for editing the code of computations, but also for reprogramming and extending the notebook itself (Codestrates will be discussed in more detail in Section 3.2.2).

The laboratory notebook

Computational notebooks are related to the classic form of handwritten laboratory notebooks. Klokmose and Zander (2010) investigated the role of laboratory notebooks for physicists. Their results show that laboratory notebooks were — and still are — used for all kinds of tasks, for instance, the logging and structuring of experimental results, the analysis of intermediate results, or the documentation of considerations for future experiments or mistakes of the past (Klokmose and Zander, 2010). They further report that these tasks are not only fulfilled individually but also by groups of people using the same shared physical notebook. Thus notebooks can also act as an instrument of communication. While the laboratory notebook fulfills the various tasks of multiple people, its "inscriptions do not seem tar-

geted towards creating a final product, like the inscriptions of a letter or a paper are" (Klokmose and Zander, 2010).

Personal, exploratory, and messy

The idea that laboratory notebooks are not the final product but more of a space for exploration and documentation is mirrored in their digital counterparts. In their research on computational notebooks Rule et al. (2018) discuss how researchers and data scientists use computational notebooks or, more precisely, Jupyter Notebook. Their analysis of over 1 million computational notebooks on GitHub[7] shows that there is "a tension between exploring data and explaining process and how this hinders construction and sharing of computational notebooks" (Rule et al., 2018). This is, for example, noticeable when looking into the number of text cells of the analyzed notebooks: 27.6 % of the notebooks do not have a single text cell in it (see Figure 5). Kery et al. (2018) reveal that users do indeed create narrative structures during the process of exploration, however, instead of using text cells, they use the structure of the code cells themselves. Computational notebooks — same as analog laboratory notebooks — are mainly used for exploration and seldom to create a final product that features an extensive explanation and documentation of what it does.

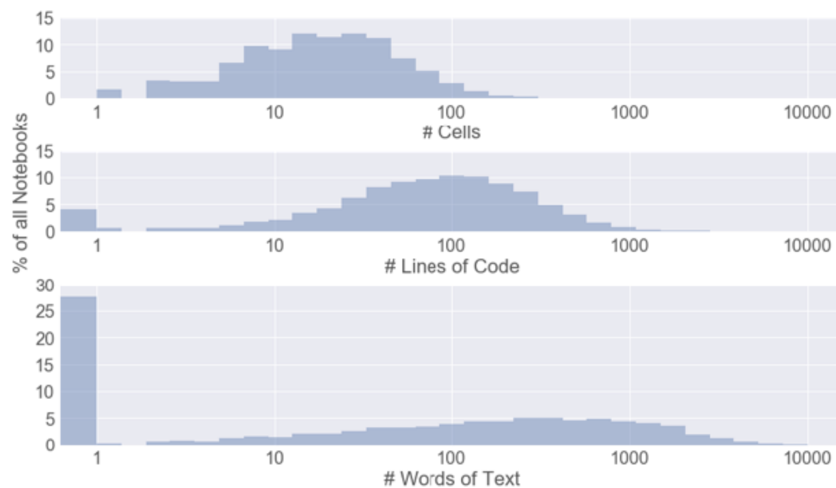


Figure 5: Computational notebook analysis by Rule et al. "Notebook length as measured by cells, lines of code, and words of markdown. While only 2.2 % of all notebooks had no code, 27.6 % had no text." (Rule et al., 2018)

Use in education

The use of computational notebooks for teaching or learning activities has been explored in areas like artificial intelligence or chemistry. O'Hara et al. (2015) exploited Jupyter to teach artificial intelligence in "easy-to-use interfaces" (O'Hara et al., 2015). Srnec et al. (2016) explained reciprocal space to undergraduate students by providing a computational notebook that converts real space vectors into recipro-

7 GitHub: https://github.com/ (accessed November 15, 2018)

cal space vectors. Both valued the possibility to combine theory and materials from a lecture with code and executable equations. Other authors looked into the teaching of computing (Wilson et al., 2014) and how notebooks could be used to create and "autograde" assignments (Hamrick, 2016).

3.1.2 *Code Playgrounds*

Prototypes and experiments

Code playgrounds are web platforms where web developers can explore and test code. Fiala et al. (2016) name them "pastebin-style application" and describe them as "designed to store code experiments and examples, and are often used as supplements to programming community forums such as Stack Overflow" (Fiala et al., 2016). Platforms like CodePen[8] or JSFiddle[9] allow developers to tinker with code in the browser, quickly build prototypes, and share these with others. However, the application state of these prototypes is transient, meaning that it does not persist beyond page reloads or multiple devices. This limitation makes code playgrounds primarily a tool for exploring and testing. Building functional applications still require developers to transfer code from a playground into their software development tool.

CodeCircle

CodeCircle by Fiala et al. (2016) is a collaborative coding web platform that combines the features of simple code playgrounds like CodePen or JSFiddle, live coding environment such as GLSL Sandbox[10], and a web-based IDE like AWS Cloud9[11]. Fiala et al. (2016) describe it as "an environment for real time social coding." It uses ShareDB[12] to record changes in a database and to enable collaboration and sharing (Webstrates also uses ShareDB, Webstrates will be covered in more detail in Section 3.2.1). The user interface is divided into a code editor, a comments pane, and a live preview behind the semi-transparent editor (see Figure 6).

Codestrates

Codestrates, as already described in the previous section Computational Notebooks, also features traits of code playgrounds. New codestrates can be created, and existing codestrates can be copied by just opening a link. This allows for prototyping just as in code playgrounds, however, holding "the potential to become usable applications by persisting their states" (Rädle et al., 2017).

8 CodePen: `https://codepen.io/` (accessed November 15, 2018)
9 JSFiddle: `https://jsfiddle.net/` (accessed November 15, 2018)
10 GLSL Sandbox: `http://glslsandbox.com/` (accessed November 15, 2018)
11 AWS Cloud9: `https://aws.amazon.com/cloud9/` (accessed November 15, 2018)
12 ShareDB: `https://github.com/share/sharedb/` (accessed November 15, 2018)

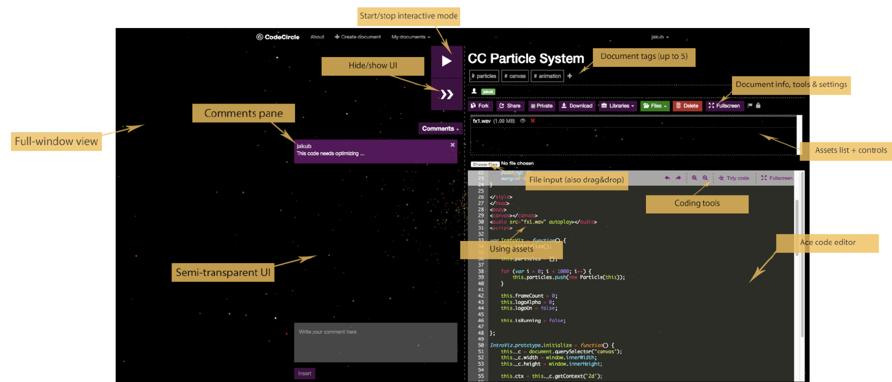


Figure 6: Screenshot of the CodeCircle user interface. (Fiala et al., 2016)

3.1.3 *Online Office Suites*

Web-based applications

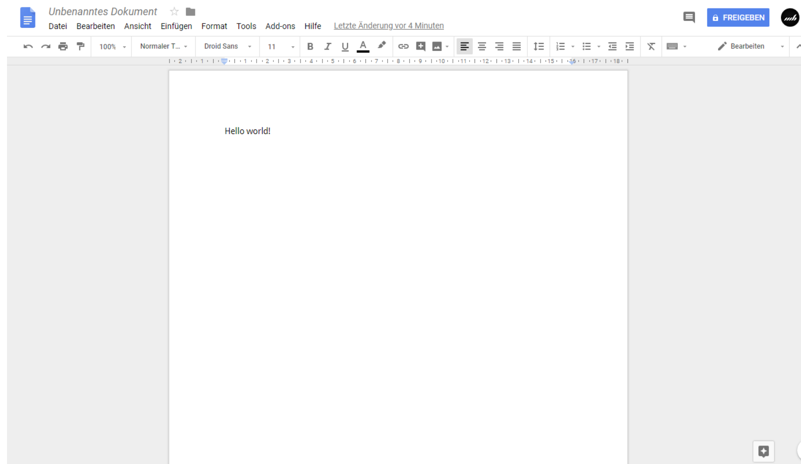
Web applications and—even more—progressive web apps[13] allow users to access applications right in their web browser without the need of an installation or setup. One popular type of web application is the online office suite. Similar to traditional office suites like Microsoft Office[14], online office suites offer users tools to write documents, create spreadsheets or presentations. However, due to their web-based nature, they allow for real-time collaboration amongst multiple users and makes sharing of documents easier.

One of the first platforms that established itself was Google Docs (see Figure 7a). After it, also Microsoft's Office Online[15] provided users with a similar tool that works hand in hand with its offline counterpart (see Figure 7b). Lastly, Dropbox Paper[16] was introduced by Dropbox (see Figure 7c). Dropbox Paper thereby focusses on a very simplistic user interface without showing the usual page layout as Google Docs and Office Online do it. However, Dropbox Paper allows users to export their documents as Word or PDF files and to present them as a slideshow right within the web app.

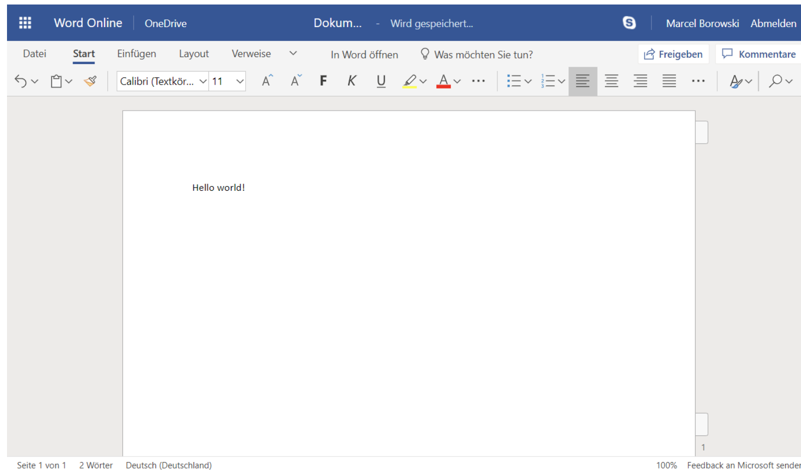
Real-time collaboration

Probably the biggest advantage of these platforms is their capability to support real-time collaboration. All three platforms support users with many features like easy sharing via a URL (Uniform Resource Locator), showing which user is currently working on the document and also showing what they are typing or what page or slide they are looking at. Tools like an integrated chat in a sidebar or the possibility to add comments to parts of a document, say a paragraph, slide or image, even further encourage collaboration—both synchronous and asynchronous.

13 Progressive Web Apps: https://developers.google.com/web/progressive-web-apps/ (accessed November 15, 2018)
14 Microsoft Office: https://products.office.com/ (accessed November 15, 2018)
15 Office Online: https://products.office.com/office-online/ (accessed November 15, 2018)
16 Dropbox Paper: https://www.dropbox.com/paper/ (accessed November 15, 2018)



(a) Screenshot of Google Docs.



(b) Screenshot of Office Online.

(c) Screenshot of Dropbox Paper.

Figure 7: Examples of online office suites.

Collaborative writing

Various researchers in the past have studied the effects of collaborative writing on platforms like Google Docs. For instance, the work of Suwantarathip and Wichadee (2014) concludes that students of a language course working collaboratively with Google Docs on writing assignments achieved better scores than students who worked in a face-to-face classroom. They suggest that working with Google Docs encouraged students to collaborate more and to assist each other in their writing assignments by supporting "students to help one another in learning without restriction of time and space" (Suwantarathip and Wichadee, 2014). Although the generalizability of the study is limited by the low number of students taking part in the study, they come to the conclusion that "students can gain a lot of benefits of blended learning when technology is applied more in language classrooms" (Suwantarathip and Wichadee, 2014).

Another work by Zhou et al. (2012) comes to a similar conclusion by stating, that "Google Docs was a useful tool for collaborative writing" (Zhou et al., 2012). Zhou et al. thereby focused on out-of-class collaboration. Their results indicate that students use fewer communication tools when being introduced to Google Docs: they used less Facebook[17] and text messaging, and instead used the comment and chat tools provided by Google Docs. At the same time, they found out that there are still challenges in using such platforms when users are new to them or not adequately instructed. Therefore they point out that instructors need to "provide detailed in-class demonstrations with specific examples" in order to "prevent these problems from precluding successful use of Google Docs" (Zhou et al., 2012).

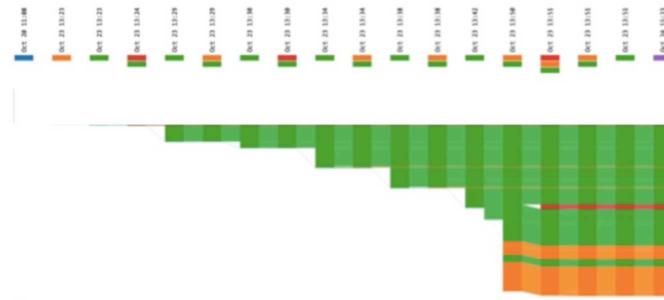
Styles of collaboration

Yim et al. (2017) analyzed the use of Google Docs for collaborative writing activities. Their study revealed four different styles for synchronous collaboration that are characterized by different ways how work is divided across group members (see Figure 8). Olson et al. (2017) further investigated Google Docs regarding synchronous and asynchronous collaborative activities and showed that group members could take on different roles during the process, collaboration can unfold in various ways, and that a shared document not only serves as a final product but also as a platform to share content and coordinate activities.

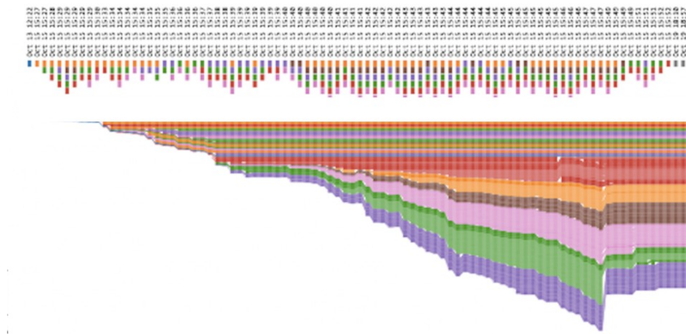
3.2 FRAMEWORKS

In the following, the platforms Webstrates and Codestrates will be discussed in more detail. Both their concepts and technical implementation will be discussed to present a clear picture of how they function.

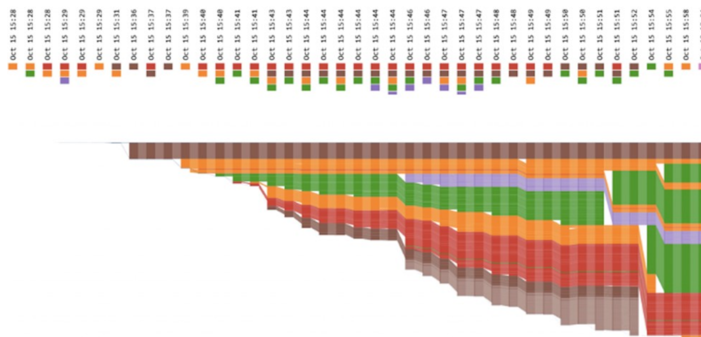
17 Facebook: `https://www.facebook.com/` (accessed November 15, 2018)



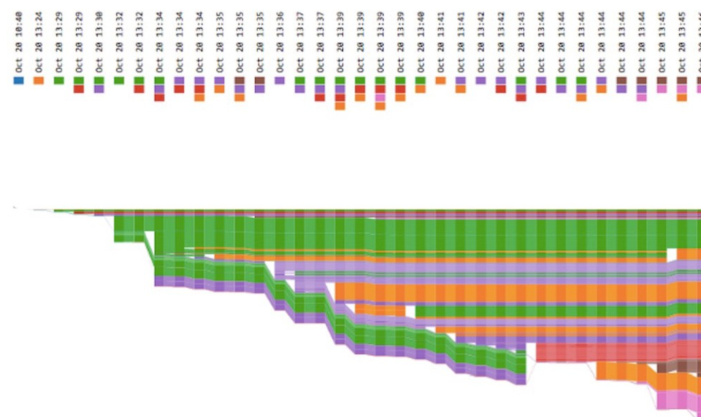
(a) Illustration of the *Main Writer* style.



(b) Illustration of the *Divide and Conquer* style.



(c) Illustration of the *Cooperative Revision* style.



(d) Illustration of the *Synchronous Hands-on* style.

Figure 8: Styles of collaborative writing. (Yim et al., 2017)

3.2.1 *Webstrates*

Shareable Dynamic Media

In their work Webstrates, Klokmose et al. (2015) present their vision of *shareable dynamic media*. Shareable dynamic media is characterized by three key properties:

MALLEABILITY: Users are able to modify the tools, user interface, or documents of a system in order to fit it to their own needs. Apart from modifying the range of functionality, it is likewise possible to extend it. So users can — on their own — adjust the system to their preferences. This functionality could be, for example, a toolbar, which the user can customize to his needs or extend depending on the task.

SHAREABILITY: On the one hand users have the possibility to use different kinds of data within the same document, and on the other hand, they can work on these documents collaboratively and simultaneously. As an example, it could be possible that multiple users can collaboratively work on the same document at the same time while using their own customized views on their own devices.

DISTRIBUTABILITY: The use of tools and documents works across multiple devices and documents can be distributed easily. For instance, the ability to open a document on both a desktop computer and a mobile phone.

Implementation of the concept

As a research prototype, Webstrates serves as a web-based platform for the creation of applications following the concept of shareable dynamic media. Using web technologies, Webstrates slightly — yet impactfully — changes the behavior of web pages. Changes to the Document Object Model (DOM) of web pages are synchronized and persisted on a server when using Webstrates (see Figure 9), thus transforming them into *webstrates* (web + substrates). "Substrates are software artifacts that embody content, computation and interaction, effectively blurring the distinction between documents and applications" (Klokmose et al., 2015). By synchronizing the DOM, changes to the HTML, JavaScript code, or CSS (Cascading Style Sheets) rules are instantly made available on all clients, allowing for a collaborative style of working.

Building on top of web technologies enables Webstrates to be used across different types of devices and operating systems. This brings computation closer to the vision of ubiquitous computing, where computation is independent of devices, and switching devices is possible in a seamless way (cf. Section 2.4).

Transcluding webstrates

Another concept of use in Webstrates is *transclusion*. Transclusion describes the process of using the content of one webstrate within another webstrate, thus transcluding it within the other. For example,

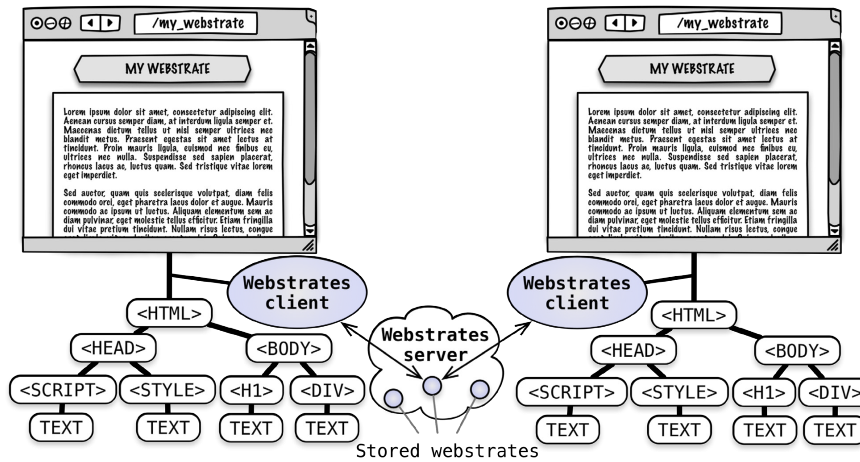


Figure 9: Synchronization of the DOM in Webstrates. Every DOM element is synchronized and stored on a Webstrates server. (Klokmose et al., 2015)

one could include the same SVG (Scalable Vector Graphics) graphic both in an SVG editor webstrate as well as in a text document webstrate. This would allow the user to edit the graphic in the editor while instantly seeing the changes in the text document. The SVG editor would be used as an instrument to alter the domain object of a SVG graphic (cf. Section 2.3).

Synchronization and persistence

In order to synchronize the DOM, Webstrates uses the ShareDB library. The library allows to edit webstrates concurrently on a server in JSON (JavaScript Object Notation) format. To transform the content of a webstrate into this format, Webstrates uses JsonML[18]. The documents in JsonML format are then stored within a MongoDB[19] database on the server. On the client side, Webstrates uses the MutationObserver[20] Web API (Application Programming Interface) to detect changes of the DOM and synchronize them with the server.

Transient elements

HTML elements are always persisted on the server. When users, however, want elements not to be persisted and synchronized on the server, they need to use a `<transient>` element[21]. Transient elements and all their children in the DOM are not persisted on the server. They are especially useful for elements of an application, which are not shared across multiple users. One example for this would be dialogs that show information to one specific user only. Besides transient elements, there are also transient HTML attributes which are not persisted. Every attribute name starting with `transient-` will not be persisted on the server.

18 JsonML: http://www.jsonml.org/ (accessed November 15, 2018)
19 MongoDB: https://www.mongodb.com/ (accessed November 15, 2018)
20 MutationObserver Web API: https://developer.mozilla.org/en-US/docs/Web/API/MutationObserver/ (accessed November 15, 2018)
21 Transient elements: https://webstrates.github.io/userguide/api/transient.html (accessed November 15, 2018)

Versioning

When performing changes on a webstrate, every single change is stored and versioned by Webstrates. A webstrate version is represented by a simple integer number. It starts with the version `"v": 0` and then increments with every change. This process is fine-grained so that the version history even stores single keystrokes. Because of that, version numbers increase fast when developing, which makes it hard to find specific states of a webstrate in the version history. To counter this problem, Webstrates allows to tag webstrates. By tagging a webstrate, a user defines a label that points to a specific version of a webstrate — similar to how tagging works in a Version Control System (VCS). These tags can be accessed as a JSON list of tag-version pairs, which makes it easier to go back to specific versions of a webstrate and restore them.

Webstrates API

Aside from the synchronization of the DOM, Webstrates also offers users some APIs such as assets, signaling or messages[22]. These allow, for example, to send signals or messages to other clients which are not reflected in the DOM. Some functions are also available in an HTTP (Hypertext Transfer Protocol) API[23], enabling users to view tags or restore webstrates without any programming.

Authentication using GitHub

Webstrates provides an authentication functionality, which allows users to authenticate themselves in webstrates by using GitHub. This makes it possible to grant or prevent access to a webstrate depending on the user's access rights. Permissions are set on a document level — for each webstrate — and can be set to either *write*, *read* or *no access*. It also allows to send messages to specific users and to store settings depending on the current user.

3.2.2 *Codestrates*

Developing with literate computing

With Codestrates, Rädle et al. present a platform which enables users to develop applications within a webstrate (Rädle et al., 2017). Codestrates thereby builds on top of Webstrates, and thus a codestrate is also always a webstrate. As mentioned above, the platform follows the idea of literate computing (cf. Section 2.6). This approach further reduces the gap between the development and use of an application.

Paragraphs and sections

A codestrate consists of paragraphs. A paragraph is an HTML element, which contains one of four types of content. There are body, code, style, and data paragraphs (see Figure 10 and Figure 11).

BODY PARAGRAPHS: A body paragraph consists of HTML elements. Its contents can range from plain text within `<div>` elements to complex structures for applications. The content of a body paragraph can

22 Webstrates APIs: https://webstrates.github.io/userguide/api.html (accessed November 15, 2018)
23 Webstrates HTTP API: https://webstrates.github.io/userguide/http-api.html (accessed November 15, 2018)

be edited either by directly writing into the elements using the *contenteditable* Web API[24], the HTML code editor of Codestrates, or the development tools of a web browser.

CODE PARAGRAPHS: A code paragraph contains JavaScript code. The code is directly editable within Codestrates by using a JavaScript code editor[25] running in the web browser. The code of a code paragraph either can be executed manually, or it can be set to *run on load*, which means that it gets executed every time the codestrate is loaded in the web browser. Each code paragraph has a console, which can be activated to display the log output of code executions right underneath the paragraph.

STYLE PARAGRAPHS: Style paragraphs contain CSS rules. Similarly to the code of code paragraphs, the CSS code can also be edited via the editor of a web browser. CSS rules are immediately applied by the web browser so that changes are instantly visible.

DATA PARAGRAPHS: Data paragraphs contain data in JSON format. Like JavaScript and CSS this data can also be edited and viewed by means of an editor in the web browser.

Paragraph functionality

Paragraphs also have further functionality, such as the option to expand them into full screen or lock them against changes. In order to group multiple paragraphs, users are provided with sections. A section in Codestrates is a group of multiple paragraphs. Or, to put it another way, a paragraph is always part of a section. Likewise, all sections are contained in a `<div class="sections">` element (see Listing 1). For more technical details on code execution, bootstrapping, and further functions refer to Rädle et al. (2017).

System sections

Besides the content and sections created by users — the *user sections* — Codestrates also contains so-called *system sections*. System sections contain the implementation of Codestrates itself. They are composed of the same types of paragraphs as user sections and can likewise be modified by users. This allows for the extension of a codestrate's functionality and enables users to adjust a codestrate to their needs, e.g., by changing its color scheme. To turn a user section into a system section — and vice versa — a user only needs to select a toggle on a section.

Updating and sharing content

System sections are an integral part of the update mechanism of Codestrates. In order to update a codestrate, a user can *pull* system sections from another codestrate. This pulling mechanism loads the codestrate into an `<iframe>` element, i.e. the codestrate gets tran-

24 Contenteditable Web API: https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/contentEditable (accessed November 15, 2018)
25 CodeMirror: https://codemirror.net/ (accessed November 15, 2018)

scluded. Then, all system sections of the current codestrate will be overwritten by all systems sections of the transcluded codestrate, thus updating the implementation of the codestrate. A subsequent refresh of the codestrate in the web browser ensures that the new code is being executed. The user sections remain unchanged during this process.

```
1  <html>
2    <head>
3      <script id="script-main">
4        <!-- [JavaScript init script] -->
5      </script>
6      <!-- [More header elements] -->
7    </head>
8    <body>
9      <div id="sections">
10       <div class="section" name="Bootstrapping">
11         <div class="paragraph code-paragraph" name="Bootstrap
               Code">
12           <pre data-type="content" type="text/javascript" id="
                 bootstrap">
13             <!-- [JavaScript bootstrap code] -->
14           </pre>
15         </div>
16         <!-- [More paragraphs] -->
17       </div>
18       <!-- [More sections] -->
19     </div>
20     <!-- [Other body HTML] -->
21   </body>
22 </html>
```

Listing 1: General structure of a codestrate.

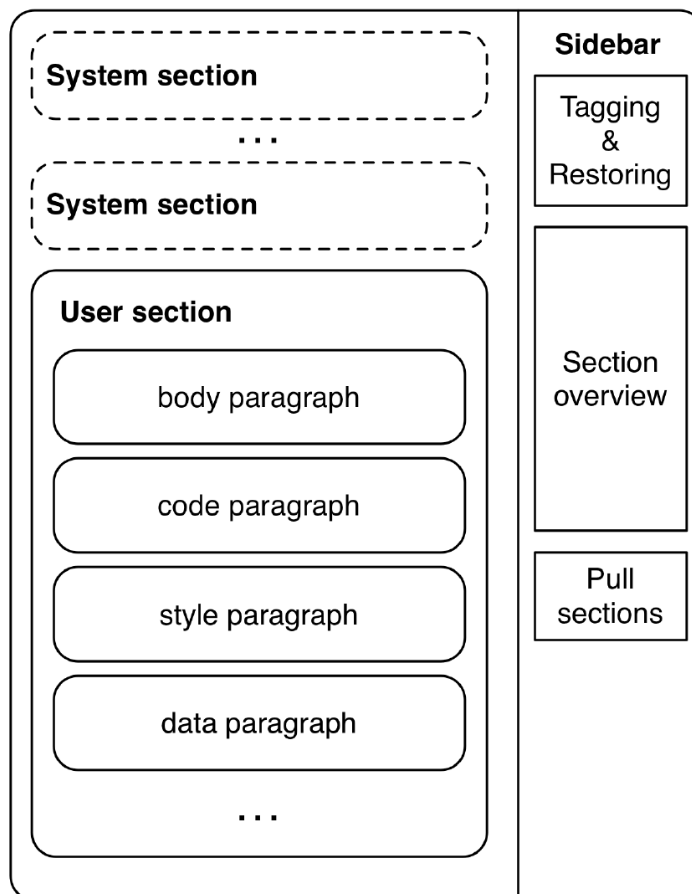


Figure 10: Schematic overview of the structure of a codestrate. On the left are the sections, content, and implementation of a codestrate. On the right side is a sidebar with additional functions. This schematic corresponds to the first version of a codestrate; it has changed throughout the master-project. (Rädle et al., 2017)

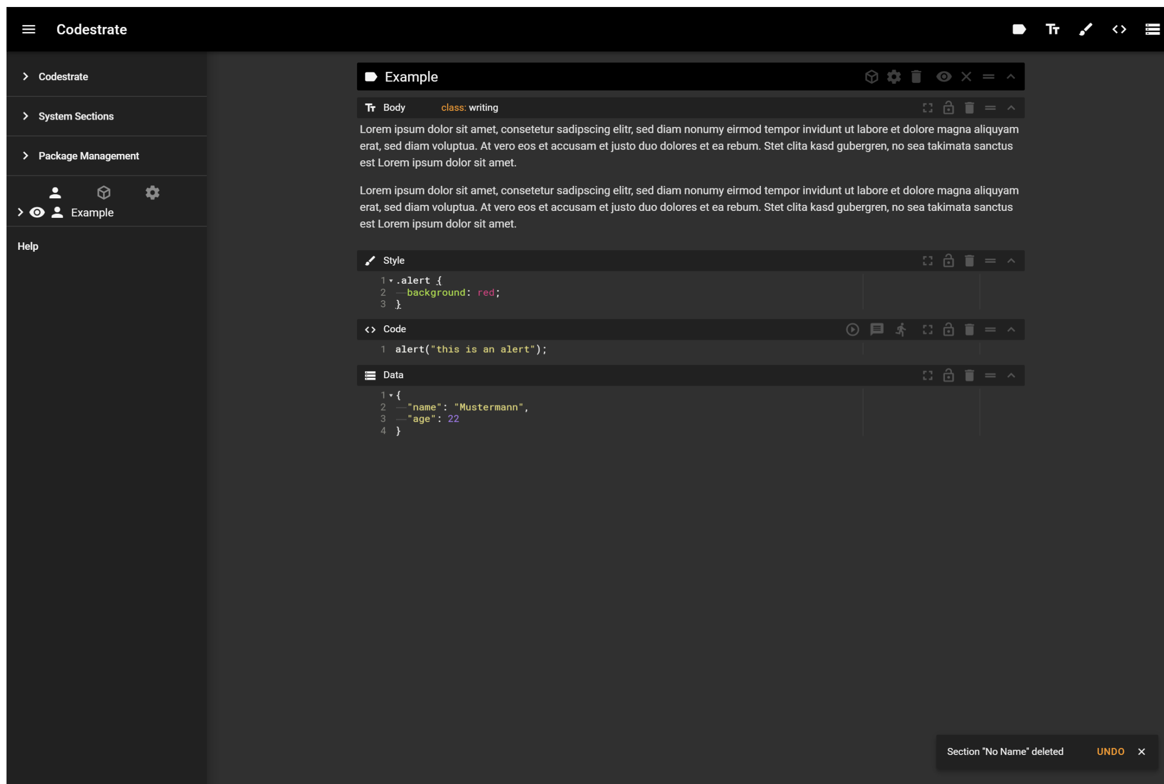


Figure 11: Screenshot of a codestrate. The screenshot shows one section with four paragraphs, the sidebar on the left is opened and shows actions and a section browser. The screenshot shows the current version of a codestrate, after the changes of the master-project.

4 SYSTEM AND SETUP

The platform used for the conduction of the study of this thesis was Codestrates and its package management. The latter being the subject of the master-project of this work — Codestrate Packages. This chapter will first introduce the fundamental concepts and components of Codestrate Packages. The section will then describe how the system was adapted to being used in the study and how the course and its tutorials were structured.

4.1 CODESTRATE PACKAGES

Just all or nothing

As described in the previous chapter, the update mechanism of Codestrates uses transclusion to pull system sections of one codestrate into another. A downside of this method is that it is only possible to pull all system sections at once. It is not possible to pull specific sections from one codestrate to another. This means that when users perform changes in a system section to adjust it to their needs, they can only update all system sections and not individual ones. This can be a problem when developing parts of a system section and wanting to update another one. This restriction also prevents users from easily sharing their user sections with others.

Features for the task at hand

Codestrate Packages advances the creation of content further from an application-centric model into a document-centric model where computation is part of documents. In contrast to the previous version of Codestrates without the package management, Codestrate Packages allows users to add new features to their documents by using *packages*. Just like in the instrumental interaction model (cf. Section 2.3), where instruments can be activated and deactivated depending on the task at hand, in Codestrate Packages, packages can be added and removed to add or remove features from a document. Therefore users are no longer confronted with a fixed feature set but can match the features to their current task at hand.

Supporting the reprogrammable nature of Codestrates, new features can also be implemented by users themselves and shared with other people without having to leave the document. In Codestrate Packages, a codestrate can act both as a document containing content and functionality and as a *package repository*, from which packages can be pulled and to which packages can be pushed to.

4.1.1 *Packages*

Extensions of functionality

The term *package* refers to functionality which can be added or removed from a codestrate. This functionality can range from themes, small changes such as shared pointers[1] or new types of paragraphs (e.g., a drawing canvas or a slide), up to fully working applications. Packages can be added or removed by users while they are using a codestrate — they do not have to decide beforehand which functionality they need but can add new functionality later.

Addition to the document — not the application

Packages can be compared with extensions in web browsers like Google Chrome[2], add-ins in Microsoft Office applications, or extensions of code editors such as Visual Studio Code[3]. However, contrary to the above examples, packages in Codestrate Packages are not added to an application but a document (see Figure 12). For instance, if a user adds an add-in to Word which enables the use of interactive maps within the document and they want to share named document with colleagues, the colleagues would need to install the same add-in to their Word application as well. Otherwise, they would not be able to use the interactive map within the document. In Codestrate Packages, however, the interactive map would be added to the codestrate document itself, which makes it possible to share the codestrate with colleagues without them needing to install the package themselves as well.

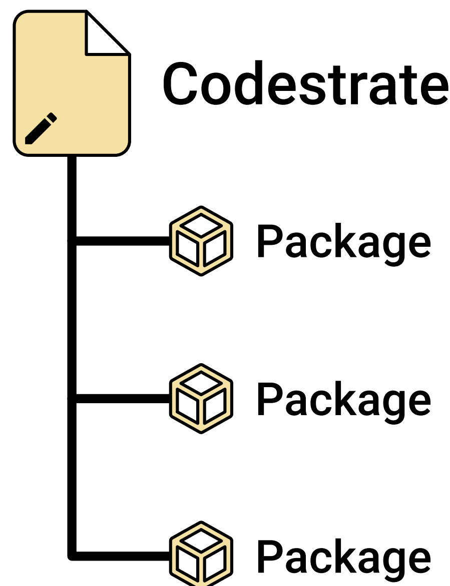


Figure 12: Packages are added to documents. A codestrate can contain any number of packages.

1 *Shared Pointers* is a package for Codestrates, which displays the pointers of other users using the same codestrate during collaboration.
2 Google Chrome: https://www.google.com/chrome/ (accessed November 15, 2018)
3 Visual Studio Code: https://code.visualstudio.com/ (accessed November 15, 2018)

Packages are sections

On the technical side, packages are realized as a new type of section. Therefore, every package is a section, and every section can be turned into a package. This is similar to how system sections work (cf. Section 3.2.2): In order to turn a user section into a system section, the HTML attribute data-type="system" is added to the <div class="section"> element. To turn a user section or system section into a package the attribute data-type="package" is added to the HTML element of the section. This also means that a section can either be a user section, a system section, or a package — yet it can only be of one of these types at the same time (see Figure 13).

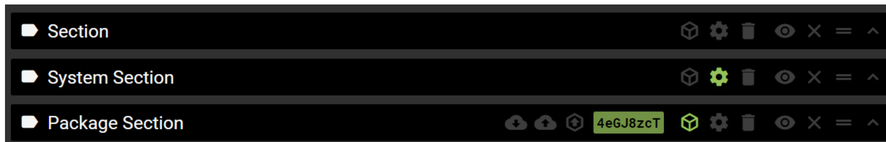


Figure 13: Three different section types. The screenshot shows three section headers. The top one is from a user section, the middle one is from a system section, and the bottom one is from a package section. Toggling a section as a package enables further functions such as ex- and import buttons.

Packages are identified by an ID. It is stored in the data-id attribute of the <div class="section"> element. Random IDs are automatically generated for all sections and paragraphs but can also be changed manually by modifying the HTML element. A package is made up of four parts: documentation, properties, the functionality, and optional assets.

DOCUMENTATION: One paragraph of each package should be the documentation. It can be any of the paragraphs but by convention, it should be the first one. The documentation is a body paragraph with the class section-documentation. In it, the usage and available APIs should be documented and illustrated with examples. The documentation serves as a reference and manual for users who want to use or possibly extend the package.

PROPERTIES: The properties are usually the second paragraph of a package. They act as a container of a package's meta information. The properties are entered into a data paragraph with the class attribute section-properties. The format of the properties is, therefore, a JSON object.

FUNCTIONALITY: The functionality a package adds to a codestrate is its key part. This can, for example, be the addition of paragraph types, new user interface elements or any other functionality. The functionality can be implemented using any combination of para-

graphs. Paragraphs inside the package section are thereby automatically part of the package.

ASSETS: In addition to using the paragraphs of a package, a user can also add files to a codestrate. These files are stored as Webstrates Assets[4], thus being part of the webstrate they were added to. If a package needs these files, the user needs to add them as assets to the properties of the package. This is necessary, as otherwise Codestrate Packages would not know which assets belong to which package. A more detailed look at how assets are copied is described in the section Package Management.

4.1.2 *Package Repositories*

Every codestrate is a repository

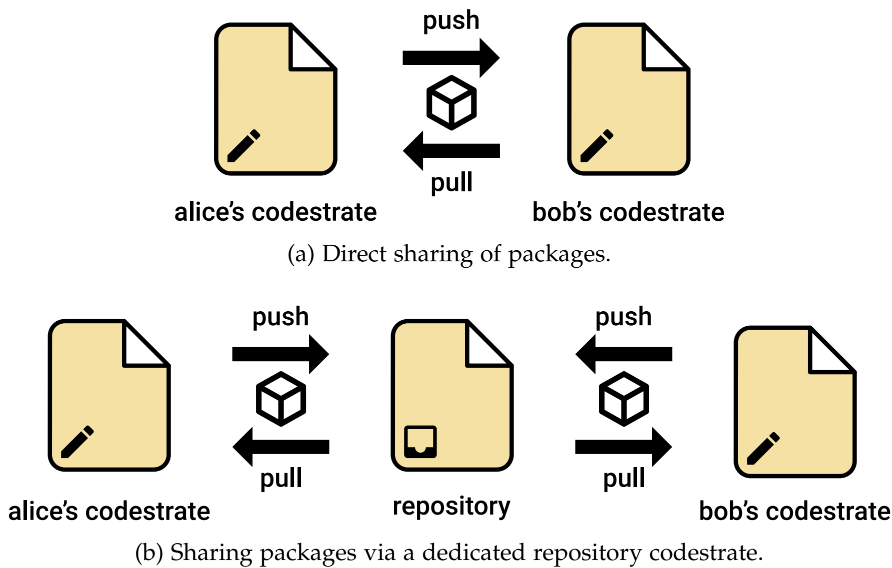
In order to pull or push packages to a codestrate, users first have to specify another codestrate that they want to pull the packages from or push to. In this scenario, the latter codestrate is called the *repository* and the former is called the *target*. A repository is, therefore, no different from a regular codestrate. Every codestrate can act both as a target at one time and as a repository at another time.

Codestrate Packages offers a manager for package repositories. It allows users to define a list of frequently used package repositories. The selected repositories are then made available in a dropdown menu when installing or pushing packages. Selecting a package repository consists of two parts: an ID and, optionally, a tag of a codestrate. Being able to select a specific tag, from which packages should be pulled, also makes it possible to retrieve older versions of packages (provided older versions have been pushed to that repository before).

Dedicated repository codestrates

By not distinguishing a regular codestrate from a repository, it is possible for users to pull packages directly from the codestrates of other users (see Figure 14a). This can, however, cause problems: Someone could be making changes to a package while, at the same time, someone else is pulling the working version of it. The user would end up pulling a nonfunctioning version of the package. Because of that, it is recommended to use a dedicated repository codestrate which contains stable versions of packages (see Figure 14b). This idea is derived from other VCSs like Git[5] or Subversion[6], where changes are made locally until a user pushes or commits them to a server.

4 Webstrates Assets: `https://webstrates.github.io/userguide/api/assets.html` (accessed November 15, 2018)
5 Git: `https://git-scm.com/` (accessed November 15, 2018)
6 Subversion: `https://subversion.apache.org/` (accessed November 15, 2018)



(a) Direct sharing of packages.

(b) Sharing packages via a dedicated repository codestrate.

Figure 14: Different types of sharing packages.

4.1.3 *Package Management*

Distributing and sharing packages

This subsection will address the functionality which manages the installation, updates, pushing, removal, export, and import of packages. The package management functionality takes the information of packages, their properties, and the list of package repositories to provide users with a simple user interface to manage the packages of their codestrates. As already mentioned, the package management includes the following six main tasks:

PACKAGE INSTALLATION: The process of selecting and installing packages on a codestrate which are not currently installed. During the installation process the package section and its assets will be pulled from the repository.

PACKAGE UPDATES: The process of selecting and updating packages installed on a codestrate. During the installation process, the package section and its assets will be pulled from the repository and thereby overwrite the previously installed version of the package. Users can choose to update individual packages or all currently installed at once.

PACKAGE PUSHING: The process of pushing packages from the target to a package repository. This is used to distribute new packages or updates to packages. It performs a similar action as an installation, but the other way round — the contents are pushed from the own codestrate to the repository.

PACKAGE REMOVAL: The process of removing installed packages from a codestrate. This process, in contrast to just deleting the respective package sections, checks whether other packages are still dependent on the ones to be removed.

PACKAGE EXPORT: The process of exporting installed packages into a ZIP file. The ZIP file contains all data of the packages, i.e. its HTML contents and its assets. This allows to share packages across different Webstrates servers, as this would otherwise be prohibited by the same-origin policy[7].

PACKAGE IMPORT: The process of importing packages from a ZIP file into a codestrate. Packages that were not previously installed are added like in the installation process and packages that were already installed become updated to the version of the imported ZIP file.

4.2 INTERACTIVE SYSTEMS COURSE

Setting of the study

The study was conducted in the introductory course *Interactive Systems* on human-computer interaction at the University of Konstanz. In order to use Codestrates and Codestrate Packages in the course as a tool, some adaptations were done. This section will first introduce the course and its structure, then it will describe how assignments in the course were structured and what topics the assignments covered, next the provided packages for development are introduced, and lastly, the workflow for the students and tutors during the course is presented.

4.2.1 *Course Description*

Theory and practice

The course takes place every other semester. It is a mandatory course in the basic studies of the computer science bachelor's program, usually taken in the fourth semester. The course is split into two parts. Part one is the lecturing part, which covers basic topics of human-computer interaction where students learn how to design "usable" interactive systems based on the textbook *Mensch-Maschine-Interaktion* by Butz and Krüger (2014). Part two is the practical part consisting of tutorials and assignments. Both parts of the course are coordinated to each other so that tasks of the weekly assignments fit the topics covered in the lecture.

Lectures

The lecture is held weekly. Topics that are covered in the lectures include human visual perception and design principles such as affordances and constraints (Norman, 2013). Students learn theoretical

7 Same-origin policy: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy (accessed November 15, 2018)

concepts and are introduced to respective examples for these concepts.

Tutorials

The tutorials accompany the weekly lecture. In them, teaching assistants present and discuss different user interface design patterns (Tidwell, 2010) with the students. These patterns then need to be applied in assignments. In the assignments, pairs of students need to develop small interactive prototypes based on these patterns and hand them in after one week of processing time. In the week thereafter, the assignments are discussed and sample solutions are presented to the students during the tutorial. Assignments in the course are graded, students need to get at least half of the points of every assignment in order to get the admission to write the exam at the end of the semester and get the credits for the course.

4.2.2 *Assignments*

Application of design patterns

The assignments of the tutorial part of the course focus on the learning of interactive systems design. A key element of the assignments is the application of various design patterns derived from the textbook *Designing Interfaces* by Tidwell (2010). For example, in one assignment students are required to implement a to-do list. In it, they should use a combination of the design patterns *Row Striping* and *Input Prompt*. The design patterns are introduced to the students in the tutorial session before the respective assignment was handed out to them.

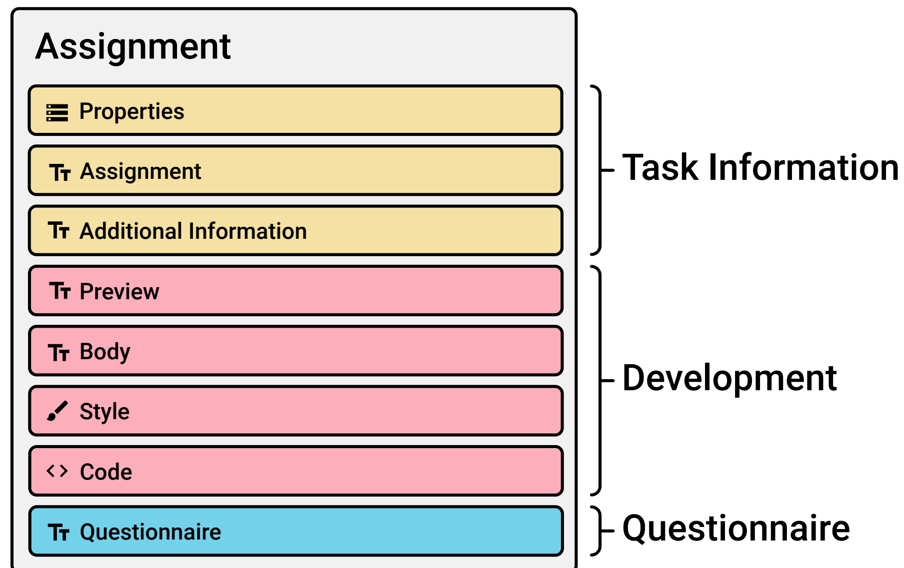
The course is open to students from other subjects as computer science. These students need to solve theoretical assignments as opposed to the computer science students. The theoretical assignments cover topics from the lecture such as the Gestalt principles.

Assignment structure

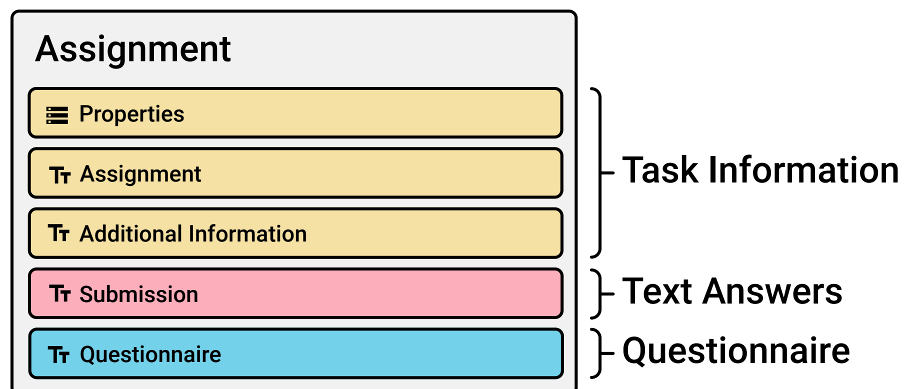
In the previous year the assignments for computer science students were handed out as PDF files and a development framework consisting of an HTML file, a CSS file, a JavaScript file, and the programming libraries jQuery[8] and Materialize[9]. The framework was provided to the students as a ZIP file. Students were free to use any code editor or IDE to process the assignments. Students handed the assignments in by uploading their submission as a ZIP file. Non-computer science students had to answer the theoretical questions in text format and return their answers as a PDF file.

In Codestrates, the assignments were modeled using a section. The computer science assignments consisted of eight paragraphs (see Figure 15a and Figure 17), the non-computer science assignments of five paragraphs (see Figure 15b). Every assignment was a package with a unique ID. The paragraphs can be split into four topics:

8 jQuery: https://jquery.com/ (accessed November 15, 2018)
9 Materialize: https://materializecss.com/ (accessed November 15, 2018)



(a) Computer science assignments contained eight paragraphs.



(b) Non-computer science assignments contained five paragraphs.

Figure 15: The structure of assignments. Each assignment is a section.

TASK INFORMATION: The first three paragraphs in the package contained descriptive information about the assignments: the *Properties* paragraph contained metadata about the assignment, the *Assignment* paragraph contained the task description and details on the specific assignment, and the *Additional Information* paragraph contained supplementary information and resources that were mostly the same in all assignments. The separation of the information into two paragraphs allowed students to hide the additional information if not needed but still keep the task descriptions visible.

DEVELOPMENT: The next four paragraphs were the development environment for the computer science students. In order to make the development in Codestrates more similar to classic web development — for instance not having to think about a persistent DOM — a small framework was implemented that allowed students to imple-

ment their applications almost like in a regular text editor: In the *Body* paragraph they could write HTML code that formed the framework of their application. The HTML was written inside a `<template>` element in order to prevent JavaScript listener to select the wrong elements (i.e. the one in the *Body* and not the one in the *Preview* paragraph). Afterwards they could style their application using CSS rules in the *Style* paragraph. Lastly, the *Code* paragraph allowed them to add interactivity to their applications using JavaScript. Changes made to the *Style* paragraph were instantly applied to their framework, changes to the *Body* or *Code* paragraphs could be applied by running the code paragraph. By doing so the HTML of the *Body* paragraph was loaded into the *Preview* paragraph and the JavaScript code was executed — thereby overwriting the old preview and old JavaScript objects (see Figure 16).

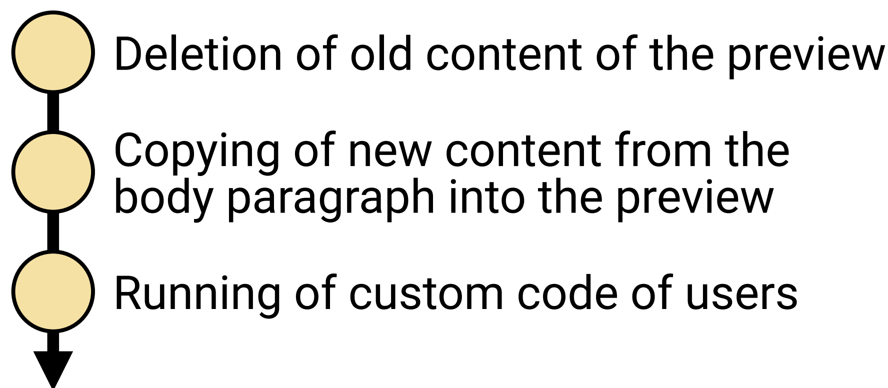


Figure 16: Execution of the code paragraph of an assignment. Using this method removes old JavaScript objects and event listeners — for instance click listeners on buttons — before running the new ones.

TEXT ANSWERS: Instead of the development part, non-computer science students were provided with a single paragraph with text fields to answer the theoretical questions of their assignments.

QUESTIONNAIRE: The last paragraph of any assignment was a questionnaire for the study. It contained multiple-choice and free text questions inside a body paragraph (the questionnaires will be covered in more detail in Section 5.2.1).

4.2.3 *Packages for Development*

A selection of packages

In order to support the students in their task of programming, there were several packages available to them (see Table 2). A recommended list of packages for the course was provided, as not all of the packages

Übungsblatt Informatik 3 — [Gruppennummer] 8QFDdLpf

Properties class: section-properties

Aufgabe class: section-documentation assignment

Interaktive Systeme — Übungsblatt Informatik 3

Allgemeines

Johannes Zagermann — johannes.zagermann@uni-konstanz.de
Philipp von Bauer — philipp.bauer@uni-konstanz.de
Lisa-Maria Mayer — lisa-maria.mayer@uni-konstanz.de
Marcel Borowski — marcel.borowski@uni-konstanz.de

Ausgabe: 29. Mai 2018
Abgabe: 04. Juni 2018 23:59 Uhr

Informationen zur Abgabe

Die Übungen zur Vorlesung *Interaktive Systeme* finden wöchentlich statt. Jeden Dienstag wird ein Übungsblatt als Codestrate Package im Übungen Repository online gestellt, das bis zum angegebenen Zeitpunkt bearbeitet und in das jeweilige Abgabe Repository gepusht werden muss. Die Lösungen müssen in Zweiergruppen eingereicht werden. Weitere Hinweise und Richtlichen zur Abgabe finden Sie im Paragraphen *Anhang*.

Aufgabe: Eine einfache To Do - Liste (10 Punkte)

Patterns

- **Animated Transitions**
  http://proquest.tech.safaribooksonline.de/9781449379711/animated_transition_html
- **Input Prompt**
  http://proquest.tech.safaribooksonline.de/9781449379711/id867095# input_prompt
- **Row Striping**
  http://proquest.tech.safaribooksonline.de/9781449379711/row_striping_html

Aufgabenstellung

Ziel dieser Übung ist es ein kleines Interface für eine einfache ToDo Liste zu erstellen. Der Nutzer soll in ein Eingabefeld einen Text eingeben können und diesen dann zu der Liste hinzufügen können. Der Fokus hier soll auf den Animationen der Listeneinträge liegen, sowohl beim Hinzufügen als auch beim Entfernen.

Anforderungen und Punkte

- (2 Punkte) Der Nutzer kann mit Hilfe eines Text-Eingabefelds (*Input Prompt*) und eines Buttons neue Einträge der ToDo Liste anfügen.
- (2 Punkte) Ein einzelner Listeneintrag soll aus dem eingegeben Text bestehen und einem Button zum Löschen des Eintrags.
  - Leere Einträge sind nicht erlaubt
- (2 Punkte) Die Listeneinträge sollen alternierende Hintergrundfarben besitzen (*Row Striping*).
  - Auch nach Aktualisierung der Liste (Hinzufügen oder Entfernen von Einträgen) sollte das *Row Striping* aktualisiert werden.
- (4 Punkte) Das Hinzufügen und Entfernen eines Listeneintrags wird durch eine Animation visuell unterstützt (*Animated Transition*).

Hinweise

- Eine Hinzufügen Animation wäre z.B., dass der Eintrag kurz aufleuchtet.
- Beim Entfernen eines Eintrags in der Mitte der Liste sollte beachtet werden, dass im Nachhinein die alternierenden Farben noch stimmen.

Anhang class: writing assignment

Anhang

Richtlinien zur Abgabe

Um die Korrektur zu unterstützen, bitten wir Sie Folgendes bei der Abgabe der Lösungen zu beachten:

- Fügen Sie im Namen des Übungsblattes ihre Gruppennummer ein.
- Fügen Sie im Paragraphen *Properties* alle von Ihnen verwendeten Assets (Bilder, Libraries, etc.) und Dependencies (jQuery, Materialize) ein. Beispiel:

```
...
assets: [
  "example.jpg",
  "example.png
],
dependencies: [
  {
    "id": "nWrE48mS",
    "name": "Materialize"
  }
],
...
```

- Beantworten Sie die Fragen der Studie.
  - Das ist verpflichtend. Abgaben bei denen der Abschnitt Studie nicht oder nur teilweise ausgefüllt wurde werden mit 0 Punkten bewertet.
- Pushen Sie die bearbeitete Aufgabe in das Abgabe Repository Ihrer Gruppe vor Ablauf der Abgabefrist.
  - Die letzte vor der Deadline gepushte Version wird bewertet.
  - Bearbeiten Sie das Übungsblatt nicht direkt in dem Abgabe Repository, nur über den *Push Package* Dialog abgegebene Lösungen werden bewertet.
- Schreiben Sie das HTML in das `<template class="assignment-X-template>` Element des *Body* Paragraphen indem Sie den HTML Editor in der Snackbar (unterer Bildschirmrand) nutzen.
- Schreiben Sie das CSS in den *Style* Paragraphen.
- Schreiben Sie den JavaScript Code in den *Code* Paragraphen.

Generelle Anmerkungen zur Aufgabe

- Wir testen Ihre Abgaben mit der aktuellesten Version von Google Chrome.
  - Testen Sie Ihren Code daher bitte damit.
  - Browser-Kompatibilität spielt keine Rolle.
- Die Library *jQuery* und das Framework *Materialize* stehen als Packages auf dem Packages Repository zur Verfügung und dürfen verwendet werden, sofern es in der Aufgabenstellung nicht anders angegeben ist.
- Um Ihre Abgabe zu prüfen (z.B. ob alle Assets gepusht wurden) können Sie schlicht einen neuen codestrate erstellen und ihre eingereichte Lösung aus Ihrem Abgabe Repository pullen und prüfen.

Figure 17: Screenshot of a computer science assignment.

Hinweise zu Anforderungen und Bewertung

Wir erwarten, dass Sie die Beschreibungen der Patterns aus dem Buch *Designing Interfaces* für die Aufgabenstellung angemessen nutzen. D.h. nicht jedes kleinste Detail ist immer unter den Anforderungen der jeweiligen Aufgabenstellung zu finden. Im Buch unter dem Abschnitt *How* werden einige Hinweise gegeben, die bei der Umsetzung behilflich sind.

Ein Beispiel:

- **Aufgabe**: „Nutzen Sie das *Loading Indicators Pattern*."
- **Anforderung**: „Wenn neue Bilder geladen werden, erhält der Nutzer Rückmeldung über den Zustand durch Loading Indicators."
- **Erwartung**: Wir erwarten hier ohne dies explizit zu erwähnen z.B. dass der Indikator animiert ist.

Nützliche Links

- **Designing Interfaces (Buch)** (Über das Uni-Netz verfügbar) http://proquest.tech.safaribooksonline.de/9781449379711
- **Designing Interfaces (Website)** http://designinginterfaces.com/
- **Pattern Libraries**
  - http://ui-patterns.com/
  - http://welie.com/index.php
  - http://patternry.com/patterns/
- **Farben und Icons/Bilder**
  - https://material.io/icons/
  - https://material.io/color/
  - https://thenounproject.com/
  - https://color.adobe.com/de/create/color-wheel/

Preview #assignment-3

Hello World!
Edit the Body Template using the HTML editor in the snackbar at the bottom.

Body

Style

```
1 #assignment-3 .example {
2   background: white;
3   color: black;
4 }
5
6 /* Style here */
```

Code

```
1 Codestrate.loadTemplate("assignment-3-template", "assignment-3");
2
3 /*****************************************************************************/
4
5 // Code here
```

Studie #assignment-3-study class: writing assignment

Studie

Im Rahmen der Studie zu meinem Master-Projekt möchte ich Sie bitten die folgenden Fragen kurz zu beantworten.

Vielen Dank.

Arbeitsweise

Wie sicher fühlen Sie sich im Umgang mit dem Package Management?

Sehr sicher ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 Sehr unsicher

Hat das Package Management Ihre Arbeitsweise (zum Beispiel gegenüber anderen Entwicklungsumgebungen) beeinflusst? Falls ja, inwiefern?

Ihre Antwort...

Welchen prozentualen zeitlichen Anteil hat das Package Management bei der Lösung des Übungsblatts etwa beansprucht?

Ihre Antwort...

Kollaboration

Wo haben Sie räumlich gesehen das Übungsblatt bearbeitet?

- ○ Zusammen im selben Raum
- ○ Räumlich getrennt

Wann haben Sie zeitlich gesehen das Übungsblatt bearbeitet?

- ○ Jedes Gruppenmitglied zu unterschiedlichen Zeiten
- ○ Beide Gruppenmitglieder zur selben Zeit

Auf wie vielen Codestrates haben Sie gearbeitet?

- ○ Ein Codestrate pro Gruppe
- ○ Jeweils ein Codestrate pro Gruppenmitglied
- ○ Mehrere Codestrates

Haben Sie die Aufgaben aufgeteilt?

- ○ Alle Aufgabenteile wurden zusammen bearbeitet
- ○ Aufgaben wurden nach Bestandteilen aufgeteilt (HTML, CSS, JavaScript)
- ○ Aufgaben wurden nach Themen aufgeteilt (z.B. Formularvalidierung und Auswertung)

Falls mehr als ein Codestrate verwendet wurde, wie wurden die Ergebnisse zusammengeführt?

- ○ Das Übungsblatt wurde über das Abgabe Repository geteilt
- ○ Der Code wurde wurde direkt vom einen in den anderen Codestrate kopiert

Allgemein

Hatten Sie Probleme bei der Verwendung von Codestrates oder dem Package Management?

Ihre Antwort...

Haben Sie Verbesserungsvorschläge oder Wünsche für Codestrates oder das Package Management?

Ihre Antwort...

Figure 17: Screenshot of a computer science assignment.

available[10] were stable and made sense for the students to use. The packages were provided in a distinct repository codestrate. In addition to that, students were also allowed to try out all other packages that were available.

Programming libraries as packages

There were also two packages that were implemented especially for the course: *jQuery* and *Materialize*. Installing these packages allowed students to use functions and styles of the same-named JavaScript libraries jQuery and Materialize. The packages were implemented in a way that allowed users to use the libraries right after installing the packages without the need to further import script or style files.

4.2.4 *Workflow*

Assignment preparation

The assignments were prepared by the teaching assistants in the week before handing them out (see Figure 18). They used a template provided by the conductors of the study to create new assignments. To prevent students from accessing the assignments early, the teaching assistants used a repository that was only accessible to them but not to the students using the permissions functionality of Webstrates — this type of repository was called *private*. After the weekly tutorial, teaching assistants pushed the assignments into another repository that was accessible to students — this type of repository was called *public*. The access of the students, however, was restricted to the *read* permission, i.e. they could not modify or delete the assignments in the repository. This meant that every teaching assistant and every student participating in the course needed to authenticate themselves in the Webstrates server using their GitHub account (cf. Section 3.2.1).

Assignment processing

To modify and process the assignments, students first needed to create a new codestrate using a link on the course website or visit a codestrate they created previously. The students then could pull the assignment as a package from the public repository into their own codestrate. After processing the assignment, students needed to push the assignment into their own *submission repository*, a repository that was only accessible to the pair of students and the teaching assistants. Using the submission repository also worked the other way round: students could pull their submission out of the submission repository into one of their codestrates and continue working on them. During the processing time, students were allowed to create any number of codestrates and pull and push their submission as often as they want (the flash drive contains a video illustrating this process; cf. Appendix A).

Assignment correction

After the processing time was over, the teaching assistants pulled the submissions of the pairs of students into their own correction codestrates to which only the teaching assistants had access to. Each

10 Codestrate Packages: https://github.com/Webstrates/Codestrate-Packages (accessed November 15, 2018)

NAME	DESCRIPTION
PERMISSIONS	This package allows to change the permissions of a codestrate in a dialog.
TEXT TOOLS	This package allows for rich-text editing in body paragraphs.
AVATARS	This package allows to see an avatar for every user currently online in a codestrate.
VIDEO COMMUNICATION	This package allows to start video communication with other users currently online in a codestrate using WebRTC.
SHARED POINTERS	This package allows to see the mouse pointers of other users currently online in a codestrate.
REMOTE CURSORS	This package allows to see the text cursors of other users currently online in a codestrate.
LIGHT THEME	This package inverted the color theme to display Codestrates in a bright appearance.
JQUERY	This package imported the jQuery library into a codestrate.
MATERIALIZE	This package imported the Materialize library and CSS styles into a codestrate.

Table 2: List of packages recommended to the students.

assignment was corrected and graded by the teaching assistants. After that, the graded submissions were pushed back into the submission repositories of the respective pairs of students so they could see how well they did.

Tutorial organization in Codestrates

Besides the processing of assignments, Codestrates and its package management were also used for other organizational matters of the practical part of the course. The slide decks for the tutorial part of the course were created in Codestrates using the *Presentations* package, and the sample solutions were implemented using the same development environment consisting of four paragraphs like on the assignments. Both the slide decks and sample solutions were created and prepared in private repositories, and then pushed into public repositories as packages when making them available to students. A list of the students of the course, their pairings, and an overview of the grades for each assignment and each group were also created in Codestrates.

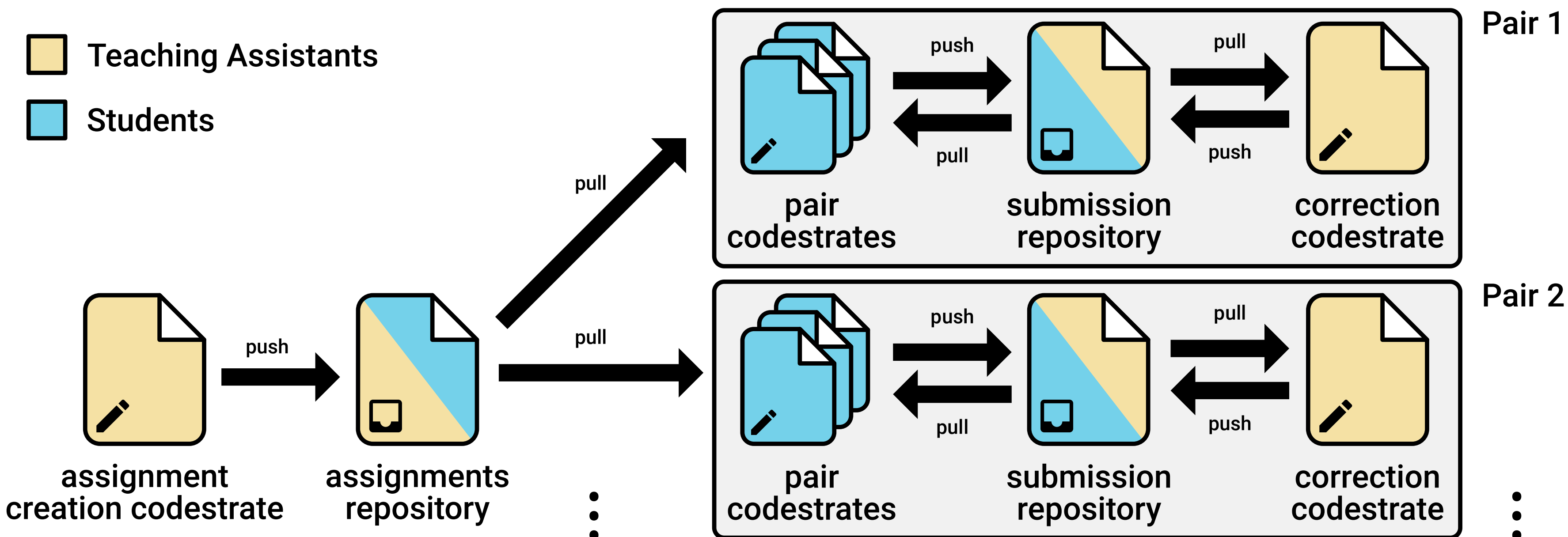


Figure 18: The workflow of distributing and submitting assignments. Teaching assistants create assignments and push them to a repository available to students. The students pull the assignments, process them in one or multiple codestrates and submit them to their own submission repository. Teaching assistants pull the submissions, correct them, and push them back to the submission repository.

5 EVALUATION

As stated in the introduction of this thesis, the three main research questions of this evaluation are the following:

RQ1 How does collaborative programming in a computational notebook unfold?

RQ2 How does the structure of computational notebooks affect programming?

RQ3 How do the malleability and extensibility of Codestrate Packages influence programming?

This chapter will first give an overview of the study, its participants, procedure, and apparatus. Next, it will present the types of data that were gathered and how the data was analyzed. Lastly, the findings of the study are described and discussed.

5.1 STUDY

The study that took place in an "in-the-wild" setting, i.e. the system was used on the devices of the students, and in their own spaces during the course and not in a controlled lab environment. The duration of the study was 13 weeks. The following subsections will provide more detailed information about the participants of the study, its procedure, and its apparatus.

5.1.1 *Participants*

Three user groups

The participants of the study were students from computer science programs, students from non-computer science programs, and the teaching assistants. Each user group used the system in a different way, providing multiple views on the experience of using the system for various tasks. The demography of the user groups is incomplete as not all students complied to take part in the demographic questionnaire.

TEACHING ASSISTANTS: In total there were three teaching assistants involved in the course. Two of the teaching assistants had experience from teaching the course the year before, while one teaching assistant participated in the course and passed it the year before. One

teaching assistant was responsible for preparing slide decks and carrying out the tutorial sessions. The other two teaching assistants held the responsibility to prepare sample solutions to the assignments, convert the assignments of the previous year into the new assignments structure (cf. Section 4.2.2), and correct the submissions of the students.

Two of the teaching assistants were male and one female. One teaching assistant was 31 years old, one was 28, and one was 23. One was in a Ph.D. program, one in a master program, and one in a bachelor program.

COMPUTER SCIENCE STUDENTS: 58 computer science students participated in the study. Working together in pairs, the students formed a total of 29 pairs. As mentioned in the previous chapter, the students solved small programming assignments and used Codestrates as their development environment.

Within the first three weeks, six out of the 29 pairs dropped the course. Reasons stated for this were varying: some groups reported that they did not have enough time to take part in the course as their other courses were more time-consuming, other groups reported that they already had passed the assignments the previous year and therefore were already admitted to write the exam in the current year. These six pairs were excluded from the data analysis.

Of the remaining 46 students (23 pairs), 17 students were male, three students were female, and 26 students did not provide this information. The average age of the 16 students that disclosed their age was 22.63 years, ranging from 19 to 29 years. 20 students were in their bachelor program, and 26 did not disclose this information. However, as the course is mandatory for students in their bachelor program, it is likely that the 26 students who did not disclose this information were also in their bachelor program. 30 students were in their fourth semester, four students were in their third semester, four were in their sixth semester, one student was in an "other" semester, and five students did not disclose this information.

NON-COMPUTER SCIENCE STUDENTS: Five students from non-computer science programs participated in the study. These students had to work on theoretical assignments and — contrary to the computer science students working in pairs — processed their assignments individually.

Three students were male, and two were female. One student was 20 years old, one student was 22 years old, one student was 27 years old, and two students did not disclose their age. Two students were in a master program, one was in a bachelor program, and two did not disclose this information. Three students were in their second semester, and two did not disclose this information.

5.1.2 *Procedure*

Initial tutorial sessions

The course took place over a period of 13 weeks (see Table 3). The initial three tutorial sessions included: (i) a general introduction of the topics, structure and procedure, and pairing of students, (ii) an introduction and recap of web technologies such as HTML, CSS, and JavaScript, and (iii) an introduction to how the system is used for both the course management and assignments. These sessions provided the students with information such as practical step-by-step guides on how to retrieve, process, and submit assignments. The introduction to web technologies compensated for differences in the participants' prior knowledge and experience. They served as a training phase for the Codestrates platform.

WEEK	DESCRIPTION	OUT	IN
1.	General introduction	-	-
2.	Introduction to web technologies	-	-
3.	Introduction to Webstrates and Codestrates	0.	-
4.	Regular tutorial session	1.	0.
5.	Regular tutorial session	2.	1.
6.	Regular tutorial session	3.	2.
7.	Regular tutorial session	4.	3.
8.	Regular tutorial session	5.	4.
9.	Regular tutorial session	6.	5.
10.	Regular tutorial session	7.	6.
11.	Regular tutorial session	8.	7.
12.	Regular tutorial session	-	8.
13.	Questions about the exam	-	-

Table 3: Overview over the period of the course. OUT: Assignment that was handed out; IN: Assignment that was handed in.

Practice assignment

At the end of the third tutorial sessions, students received an ungraded but mandatory tutorial assignment that was named the zeroth assignment. In it, the students had to go through each step of the submission process once: computer science students needed to create a codestrate, pull the assignment and a development package into it, write HTML, CSS, and JavaScript sample code, add an image as an asset, and submit the assignment to their submission repository. Instead of developing and pulling a development package, non-computer science students needed to write sample answers into the submission fields of their assignment (cf. Figure 15b). The tutorial assignment served as a dry run of the processing of an assignment and allowed

students to familiarize themselves with the submission process. This assignment was also used to resolve minor issues related to the submission process before handing out the graded assignments.

Graded assignments

Starting with the fourth tutorial session, the eight graded assignments were provided. New assignments were handed out to the students at the end of the tutorial sessions, this way they could ask questions on the tasks right away in case there were any. After one week of processing time, the assignments needed to be turned back in. In the following tutorial a sample solution of the assignment was presented and made available to the students; challenges and obstacles of the previous assignment were discussed. In total there were eight assignments, most of which unrelated to each other, with exception of assignments five and six which build on top of each other (see Table 4). The topics of the assignments were the same as the ones used the year before in the course (the flash drive contains a supplementary codestrate containing all assignments; cf. Appendix A).

5.1.3 *Apparatus*

Personal devices

During the study, all participants of all three user groups used their own personal devices. Being based on web technologies, many operating systems (including Windows, macOS, Android, and iOS) and device types (including desktop computers, notebooks, tablet, and phones) were supported. Students were recommended using Google Chrome as a web browser for processing the assignments. This is because Codestrates and the realized adaptations (cf. Chapter 4) were developed and tested using this web browser. Hence, the teaching assistants also used Chrome for the correction of the assignments and the submissions had to work using it. However, students were permitted to solve the assignments in any way that suited them, allowing for different combinations of devices and styles of collaboration, for instance, co-located or remote.

5.2 DATA ANALYSIS

Data triangulation

In the study, different types of data sources were used (see Table 5). Both qualitative and quantitative data was collected continuously during the complete period of the study. The triangulation of multiple data sources helped to strengthen findings and resolve ambiguities that persisted when only a single source would have been used. The used data sources varied depending on the user group and their tasks. Data sources include questionnaires, interviews, focus groups, and log data. The following subsections will explain each data source and how they were analyzed in more detail.

TITLE	DESCRIPTION
1. INTERACTIVE FORM	A form that used the design patterns *Structured Format* and *Input Prompt*.
2. INFINITE LIST	A list of images that can be extended infinitely by pressing a button.
3. TO-DO LIST	A to-do list that allows to add and remove elements. The design patterns *Input Prompt*, *Row Striping*, and *Animated Transition* should be used.
4. IMAGE BROWSER	An image browser that allows to browse images in three different views using the *Alternative Views* and *Module Tabs* design patterns.
5. PIZZA WIZARD PART ONE	A wizard for ordering a pizza in multiple steps. In this part the first step and the template needed to be created using the design patterns *Sequence Map* and *Diagonal Balance*.
6. PIZZA WIZARD PART TWO	In this part the other steps such as the selection of ingredients needed to be implemented using design patterns *Thumbnail Grid* and *List Builder*.
7. WEATHER DATA	An overview with multiple movable panels that display weather data of cities. The design pattern *Movable Panels* should be used.
8. RESPONSIVE ONEPAGER	A responsive page showcasing the results of the previous assignments using the *Liquid Layout* design pattern.

Table 4: List of science assignments assignments.

NAME	TA	CS	NCS
QUESTIONNAIRES			
Demographic Questionnaire	x	x	x
Weekly Questionnaires		x	x
Mid-Term and End-Term Questionnaires		x	x
INTERVIEWS AND FOCUS GROUPS			
Weekly Interviews	x		
Mid-Term Focus Group	x		
End-Term Focus Groups	x	x	
End-Term Interview			x
LOG DATA			
Log Data		x	

Table 5: Overview over the data sources. TA: Teaching assistants, CS: computer-science students, NCS: Non-computer science students.

5.2.1 *Questionnaires*

Feedback on each assignment

Over the period of the study, three types of questionnaires were used. A demographic questionnaire at the beginning of the course, questionnaires on the weekly assignments to gather feedback over time, and an extended mid-term and end-term questionnaire on the fourth and eighth assignment.

DEMOGRAPHIC QUESTIONNAIRE: At the beginning of the study, after the students had finished the zeroth practice assignment, all participants of the study were asked to fill out a demographic questionnaire (the complete questionnaire is available in Appendix B). The questionnaire included questions about the gender, age, study program, and semester of the participants. Furthermore, it investigated the current programming experience of students, including a specific section about their knowledge of web technologies. Lastly, it contained questions about the participants' project experience in collaborative projects, and their experience with collaborative web platforms such as Google Docs, Office Online, or ShareLaTeX[1]. It was created in Google Forms[2] and sent to the participants as a link in an email after the fourth tutorial session after the first graded assignment was handed out.

1 ShareLaTeX: https://www.sharelatex.com/ (accessed November 15, 2018)
2 Google Forms: https://www.google.com/forms/about/ (accessed November 15, 2018)

Although encouraged multiple times, only 20 of the 46 computer science students and three of the five non-computer science students that participated in the course filled out the questionnaire. All three teaching assistants filled out the questionnaire.

WEEKLY QUESTIONNAIRES: The students that were participating in the course were asked to fill out a short questionnaire on every assignment (cf. Figure 15). The questionnaire consisted of three parts (see Figure 19), whereas the *Collaboration* part was omitted in the questionnaires of the non-computer science students, as they worked on the assignments alone.

The purpose of the first part of the questionnaire — *Way of Working* — was to get insights into how a package-based and extensible system like Codestrates changes the way of working for students. Did they spend much time using it? How was the learning curve, i.e. did the time spent decrease and feeling of confidence increase over time? The next part, *Collaboration*, focused on the way collaboration unfolded during the processing of the assignments. The questions should give a picture of how the assignment was solved and — by asking this on every assignment — show how this changed throughout the study. Lastly, the *General* part provided participants with a space where they could give general feedback and suggestions for the platform. This should give valuable insights, on what problems still occur while using the system, and what future work on the platform should focus on.

MID-TERM AND END-TERM QUESTIONNAIRES: In the fourth and eighth assignment, two additional sections were added to the weekly questionnaire as they marked the points where half of, and all of the assignments were processed (see Figure 20). The added sections focused on the use of Codestrates and the packages. The goal was to see what other use cases students could imagine for Codestrates or modular software in general. In contrast to the other sections of the weekly questionnaire, the questions of the new sections *Codestrates* and *Packages* differed between computer science and non-computer science students.

Analysis of the answers

The results of the questionnaire were analyzed using Google Docs and Codestrates. The demographic questionnaire could be evaluated using Google Docs, as the form responses could be stored directly in a spreadsheet.

The answers to the questionnaires on the assignments — both the weekly and mid-term and end-term ones — were evaluated by using Codestrates. A small application was implemented within Codestrates and used to retrieve the answers to the questionnaires of each pair and each assignment. The retrieved answers were then exported

Way of Working

How confident do you feel in using the package management?

Very confident ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 Very unconfident

Did the package management influence your way of working (e.g., in contrast to other development environments)? If so, in what way?

Your answer...

What percental amount of the workload of the assignment did the package management took up?

Your answer...

Collaboration

Where did you work on the assignment spatially?

- ○ Together in the same room
- ○ Spatially divided

When did you work on the assignment?

- ○ Each pair partner at different times
- ○ Both pair partners at the same time

On how many codestrates did you work?

- ○ One codestrate per pair
- ○ One codestrate per pair partner
- ○ Multiple codestrates

Did you split up work?

- ○ All parts of the assignment were processed together
- ○ The assignment was split by paragraph types (HTML, CSS, JavaScript)
- ○ The assignment was split by parts of the assignment (e.g., form validation and evaluation)

If you used more than one codestrate, how did you merge your results?

- ○ The assignment was shared using the submission repository
- ○ The code was copied directly from one codestrate into another

General

Did you have problems in using Codestrates or the package management?

Your answer...

Do you have suggestions for improvement or wishes for Codestrates or the package management?

Your answer...

Figure 19: Questions of the weekly questionnaire. Screenshot of the questionnaire on an assignment. The collaboration part was only distributed to computer science students. The questionnaire was translated into English for this figure; students received it in German.

Codestrates

Did you use Codestrates to implement small applications outside of the course Interactive Systems? If so, what for example (feel free to also share a link to the respective codestrate)?

Your answer...

Can you imagine using Codestrates for future projects?

- Yes
- No

If yes, what kind of projects; if no, why not?

Your answer...

Packages

In the beginning of the course various packages were presented that should ease working in Codestrates (Remote Cursors, Avatars, ...). Additionaly there was the possibility to try out experimental packages using the *Codestrate-Packages* repository.

For what kind of tasks did you use the collaboration packages (Remote Cursors, Avatars, ...) and how often did you use them?

Your answer...

Did you use and try out any other packages from the *Codestrate-Packages* repository?

Your answer...

What benefits do you think does the use of modular and collaborative software, like Codestrate Packages offers it, entail?

Your answer...

What drawbacks do you think does the use of modular and collaborative software, like Codestrate Packages offers it, entail?

Your answer...

(a) Questions of the computer science students.

Codestrates

Did you try to implement something in Codestrates or to solve an assignment of the computer science students?

- Yes
- No

If yes, what assignment or programming task; if no, why not?

Your answer...

What benefits and drawbacks does Codestrates have compared to Google Docs or Office 365?

Your answer...

Packages

What kind of use cases can you imagine where modular software could be used outside of Codestrate Packages?

Your answer...

(b) Questions of the non-computer science students.

Figure 20: Questions of the mid-term and end-term questionnaires. Screenshot of the questionnaire on an assignment. The questionnaire was translated into English for this figure; students received it in German.

into a HTML table within the codestrate, providing information about the question, the pair, and the assignment number for each answer entry.

Next, similar answers were merged to create feedback instances. These instances, again, were then grouped by several categories to create feedback groups.

5.2.2 *Interviews and Focus Groups*

Deeper insights

During the study, interviews and focus groups were used to get more insights on how participants used the system and — more importantly — why participants used the system the way they did. Conducting interviews and focus groups also allowed to dig deeper into topics than what questionnaires could capture. In total, four types of interviews and focus groups were conducted (the complete list of topics and questions can be found in Appendix C).

WEEKLY INTERVIEWS: While the weekly questionnaires focused on the students, the weekly interviews were the counterpart for the teaching assistants. During weekly sessions, each teaching assistant was interviewed in a semi-structured interview. The interview included questions about the way of working and general problems and suggestions — just like the ones the students received (cf. Figure 19). In contrast to the collaboration part of the questionnaire, the teaching assistants were interviewed about the correction process and whether students asked them for help.

MID-TERM FOCUS GROUP: After the fourth assignment, a focus group was conducted with all three teaching assistants. The focus group started with a general introduction about the satisfaction with the system so far. The teaching assistants talked about their use of the system, the tasks they used it for, and how their learning processes developed. The next topics covered in the focus group were the linear structure of Codestrates and the modularity of Codestrate Packages, including questions about the packages the teaching assistants used and for what tasks. After discussing differences to other development environments, the focus group was concluded with questions about the use of Codestrates outside the course and possible future application areas of Codestrates.

END-TERM FOCUS GROUPS: After the last assignment was handed in, two more focus groups were conducted: another one with the three teaching assistants, and one with four computer science students from four different pairs. The focus group with the teaching assistants began with questions about literate computing and computational notebooks. After benefits and drawbacks of Codestrates

compared to IDEs were discussed, the questions focused on the capability of Codestrates to serve as a platform to hand out and hand in assignments. A final discussion over the whole course summed up the focus group.

The focus group with four computer science students was similar to the mid-term focus group with the teaching assistants: it started with the linear structure of Codestrates and its composition of sections and paragraph. After this, the questions focused on the use of packages and the understanding of the modular nature of Codestrates. Next, the students talked about their way of working, their learning curve, and how they worked on the assignments collaboratively with their pair partners. They then compared Codestrates and their way of working with it, with other IDEs and their way of working in other courses and previous projects to assess the benefits and drawbacks of it. The focus group closed with questions about the potential of Codestrates for use in future projects.

END-TERM INTERVIEW: After the last assignment, an interview with one non-computer science student was conducted. The interview began with general questions about the structure and user interface of Codestrates. Next, the use of packages was discussed. This also included questions about the understanding of the package management. The interview continued with questions about the way of working — both in Codestrates and in comparison to the way of working in other courses. After discussing the benefits and drawbacks of the system, the interview focused on whether the student also solved programming assignments and whether the platform could be imagined to be used in future projects.

Notes and recordings

As the weekly interviews were not recorded, notes were taken while interviewing the teaching assistants. The interviews varied in length depending on the teaching assistant and the current assignment. On average they took about 20 to 30 minutes.

The three focus groups and the interview with the non-computer science student were recorded — both in audio and video format. The two focus groups with the teaching assistants took roughly 90 minutes, the focus group with the computer science students and the interview with the non-computer science student took roughly 60 minutes. The recordings were used to summarize the conversations, i.e. the recordings were not transcribed by the exact wording, but by the meaning of the statements of the participants.

Similar to the answers of the questionnaires, the statements of the focus groups and interviews were merged into feedback instances and then again grouped into feedback groups.

5.2.3 *Log Data*

Twofold log data

Over the period of the study, the use of Codestrates and the package management was logged by the Webstrates server used for the course. The log data was twofold: on the one hand, the package management logged all package operations, on the other hand, Webstrates logged all operations in the DOM, i.e. all edits of the HTML, of all codestrates the participants of the study used (example log entries for both types of logging are available in Appendix D).

PACKAGE MANAGEMENT LOG DATA: The package management logged every package installation, update, pushing, deletion, export, and import. The data was logged by sending a `XMLHttpRequest` to a Node[3] application, running on the same virtual machine as the Webstrates server. The body of the request included JSON code that stored information about action. The Node application stored each request in a MongoDB database. The following data of each action was logged:

- The type of action that was performed (installation, update, pushing, deletion, export, or import).
- The timestamp when the action was performed.
- The user ID of the user performing the action.
- The users that were online on the codestrate while the action was performed on.
- The ID of the codestrate the action was performed on.
- The ID of the repository codestrate (not available for deletion, export, and import).
- The IDs of the packages that were installed, updated, pushed, removed, exported, or imported.
- The package properties (cf. Section 4.1.1) of all packages available on the codestrate the action was performed on, and, if available, all packages of the repository.

WEBSTRATES LOG DATA: Webstrates stores every single operation on each webstrate by default into an operational log in a database. Operations include every edit that is made to the DOM of a document, i.e. webstrate or codestrate. When writing code in a codestrate, every edit is automatically synchronized with the server, allowing to view the history of codestrates up to a keystroke-level. Each operation that was logged included the following information:

3 Node.js: `https://nodejs.org/` (accessed November 15, 2018)

- The timestamp when the operation was performed.
- The user ID of the user performing the operation.
- The IP address of the user performing the operation.
- Information about the operation itself.

From packages to operations

The analysis of the log data was separated into three parts. The first part of the analysis involved the log data of the package management. First, the log data was filtered: only log entries of actions that were performed during the eight weeks of processing time of the graded assignments by any of the computer science students were taken into account. Using these entries, the IDs of all codestrates that were used by the students for the processing of the assignments were retrieved.

From operations to sessions

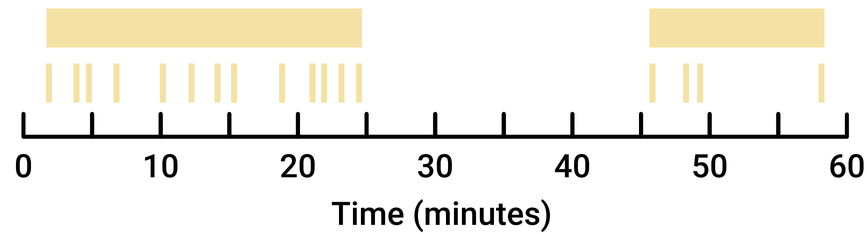
Next, the operational log of any of these codestrates was fetched from the Webstrates server's database and cached locally in JSON files. The operations were then analyzed and combined to *sessions*. A session is a timeframe within which an individual student worked continuously on a specific codestrate. For each JSON file, an algorithm iterated over all operations in that file. The timestamp of each operation was compared with the one of the previous operation. As long as the time difference was low enough, the operations were merged into the same session, when the time difference was larger than 20 minutes, the session has ended and a new session started (see Figure 21a). 20 minutes proved to be a good threshold, after empirically sampling multiple thresholds. Each session was enriched with the following information:

- The ID of the pair of students.
- The ID of the assignment.
- The user IDs of the users performing operations in the session.
- The start and end timestamps of the session.
- The ID of the codestrate of the session.
- The number of operations the session is composed of.

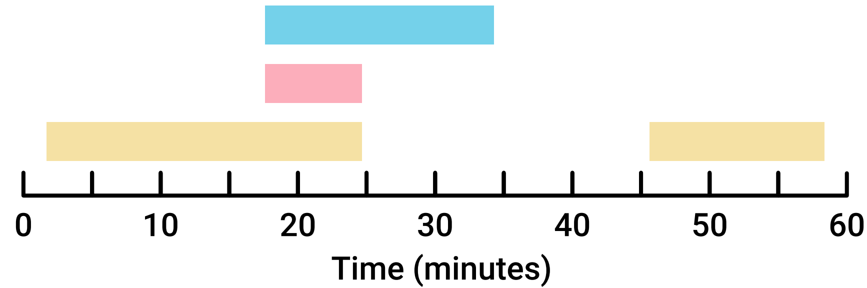
Collaborative sessions

In the last part of the analysis, the sessions of each codestrate and of each individual student were compared to generate collaborative sessions, whenever the sessions of both pair partners overlapped (see Figure 21b). Both the sessions of individual students and the collaborative sessions were stored within a single CSV (Comma-Separated Values) file containing one session per line. The CSV file, again, was then analyzed using the visualization software Tableau[4] (the Tableau project file is available in the supplementary files on the flash drive; cf. Appendix A).

4 Tableau: https://www.tableau.com/ (accessed November 15, 2018)



(a) The generation of sessions out of operations. The top row shows the generated sessions out of the operations in the bottom row. Whenever the interval between two operations was larger than 20 minutes, the sessions were split.



(b) The generation of collaborative sessions out of overlapping sessions. The top and bottom rows show sessions of two students, the middle row shows the resulting collaborative session.

Figure 21: The generation process of sessions.

5.3 FINDINGS

Three topics

In the following subsections, the findings of this study will be presented. The findings are split up into three topics — following the three research questions: the collaborative working styles of the pairs of computer science students, the influence of computational notebooks on programming compared, and the effects of the inherently reprogrammable and extensible nature of Codestrates and Codestrate Packages.

The findings of this study will focus on the teaching assistants and the computer science students as non-computer science students neither used Codestrates for programming nor worked collaboratively. Within this and the subsequent section, the pairs of computer science students will be referred to as P1 to P23, the teaching assistants as T1 to T3, and the computer science students participating in the focus group as F1 to F4. All quotations of the participants were translated from German to English.

5.3.1 *Collaborative Working Styles*

Analysis of working styles

In order to investigate the working styles of the pairs of students, the sessions of each pair and each assignment were analyzed, resulting in $23 \cdot 8 = 184$ *assignments*. Within each assignment, the students of

the pair are referred to as S1 and S2. Each assignment was manually coded for six attributes that were used to assess working styles (see Table 6; the full coding of the assignments are available in Appendix E).

NAME	DESCRIPTION
USERS	Did S1, S2, or both work on the assignment?
TIME	Did students work at the same time or at different times on the assignment?
LOCATION	Did students work co-located or remotely on the assignment?
WORKFLOW	Did students process the assignment all at once or in multiple sessions?
DOCUMENTS	Did students use one or multiple codestrates?
WORK SPLIT	Did students work on every part of the assignment together or did they split up the parts of the assignment (e.g., creating the structure of the to-do list and implementing the interactivity)?

Table 6: List of attributes used to assess working styles.

Six attributes

The *Users*, *Workflow*, and *Documents* attributes were derived from the log data, *Work Split* was derived from the questionnaire, and *Time* and *Location* were derived and matched from both the log data and the questionnaires. Each assignment was analyzed based on these attributes. Further, questionnaire data was used to reveal other influences on the collaboration between students.

Finding 1.1: Collaboration Patterns

Three groups of patterns

Pairs of students exploited several styles of collaboration during the study. After investigating the data and the attributes of each assignment, different collaboration patterns were found. The identified patterns were grouped into synchronous and asynchronous collaboration, and a working style where only one student worked in Codestrates.

SYNCHRONOUS COLLABORATION: Pairs of students worked together synchronously at the same time on 48 out of the 184 assignments. Of those 48 assignments, collaboration was co-located in 19 of them, and remote in 29 (see Figure 22a and 22b). Most of the time—in 40 assignments—, the pairs worked together on all parts of the assignment, whereas in only eight assignments students divided and conquered when working together at the same time. As described in

the section Questionnaires, splitting work was either done by working on different paragraphs (e.g., HTML, CSS, and JavaScript) or by dividing assignments into different topics (e.g., one student creating an online form and the other student adding the validation of the form afterwards). Two pairs used synchronous co-located collaboration as their primary working style and three pairs used synchronous remote collaboration (see Table 7).

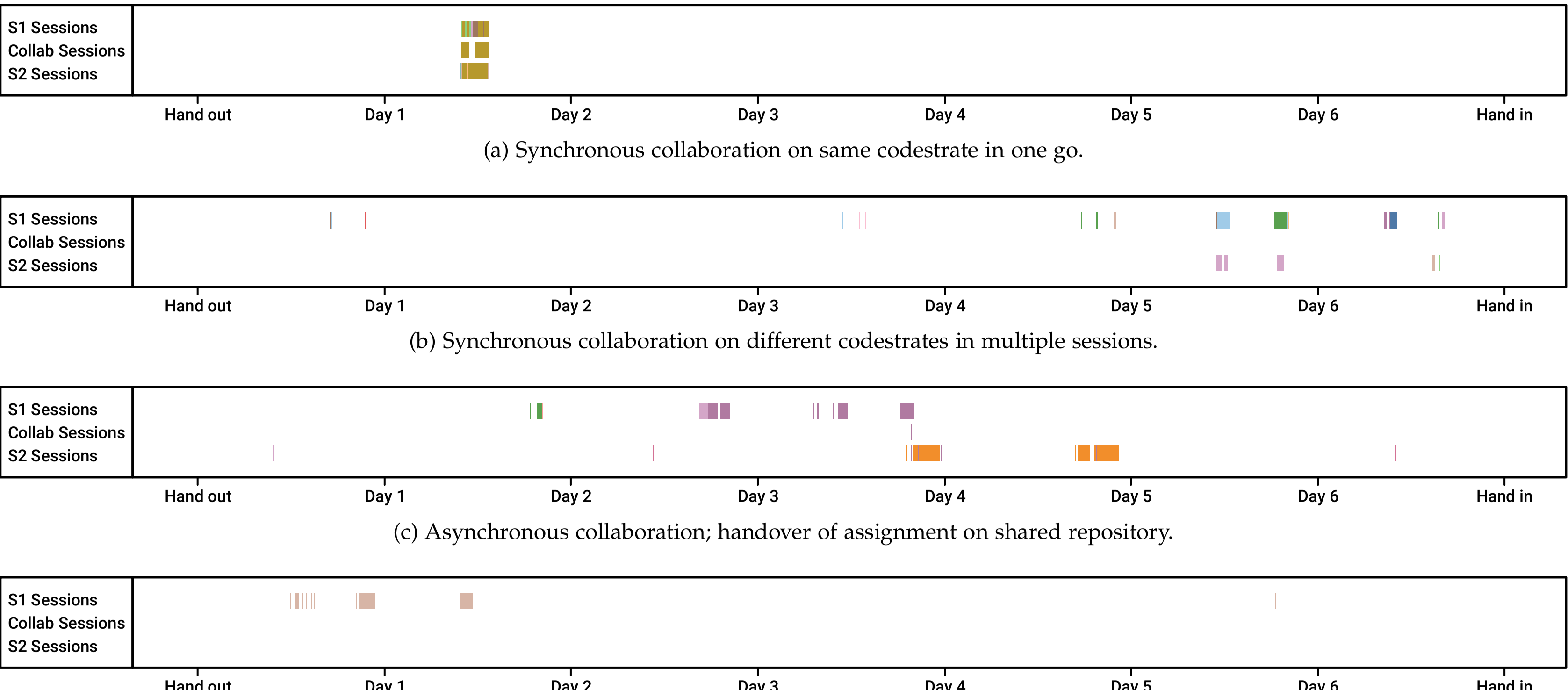
ASYNCHRONOUS COLLABORATION: In 75 of 184 assignments, students worked together asynchronously. When students were collaborating in an asynchronous manner, pairs worked remotely in all instances. Students leveraged three different ways to share intermediate results with their partner: (i) both students visited the same notebook; (ii) pairs used their submission repository as a shared repository (see Figure 22c); (iii) students shared code by copying it from one codestrate to another. Nine pairs used asynchronous collaboration as their primary working style (see Table 7).

ONLY ONE STUDENT WORKS IN CODESTRATES: In 61 of 184 assignments, only one student worked during the assignment in the computational notebook, i.e. no edits from the second student appeared in the log (see Figure 22d). However, in 23 assignments, pairs reported in the questionnaire that they did the assignment together but did not split work. This indicates that both students could have been working together in front of the same device or collaborating remotely using screen sharing. Triangulating data from questionnaires and edit logs helped to uncover behavior that is otherwise obscured, namely, who is sitting in front of the device. In six pairs, only one person worked in the computational notebook for the majority of the time (see Table 7).

Finding 1.2: Benefits and Barriers of Real-Time Synchronization

No more merge conflicts

Students reported that they valued how edits in their codestrates are synchronized instantly, as it made it easier for them to collaborate. For example, being able to work together on the same codestrate at the same time on different devices, synchronizing edits instantly. The results support this by showing that pairs used only a single codestrate per assignment in 50 of the 123 assignments, in which both students contributed to the codestrate. Compared to version control systems like Git, the students liked that there are no merge conflicts, as changes were immediately applied for each partner. However, students wished for a better change history, for example, seeing what their partner did since the last time they visited the notebook.



(a) Synchronous collaboration on same codestrate in one go.

(b) Synchronous collaboration on different codestrates in multiple sessions.

(c) Asynchronous collaboration; handover of assignment on shared repository.

(d) Only one student works in Codestrates.

Figure 22: Examples of collaborative working styles. Each row shows one assignment. Colored blocks in each row indicate sessions for $S1$, $S2$, and collaborative sessions. Each color indicates a different codestrate.

> *When I, for example, edit something in a project and my partner goes online ... [I wish] that he is notified of what I changed most recently.*
>
> — F3

Jumping content

Students also reported downsides to synchronized notebooks like Codestrates. For example, they occasionally had problems with jumping code due to another person writing in the same paragraph above. Others accidentally overwrote their partner's code. Log data shows that, for instance, P17 moved from using a shared codestrate in the first five assignments to using an individual codestrate per partner for the last three assignments — maybe as a result of problems due to the real-time synchronization.

Finding 1.3: Shareability via Links Supports Collaboration

Reduced organizational overhead

Both students and all teaching assistants reported that sharing documents via links made assignment correction and supporting students easier. For example, in previous years teaching assistants had to download zipped project archives with multiple files from the learning management platform ILIAS[5] and run the projects locally on their machine in order to correct them. Incompatible runtimes led to solutions not working correctly and there was an overhead of downloading and unpacking the solutions. This year, the teaching assistants just needed to pull the packages of the submissions of students into their codestrates in order to correct them.

5 ILIAS: https://www.ilias.de/ (accessed November 15, 2018)

WORKING STYLE	NO.	PAIRS
Synchronous co-located collaboration	6	P6
	5	P18
Synchronous remote collaboration	7	P20
	6	P10, P13
Asynchronous remote collaboration	8	P2, P5, P12
	7	P11
	6	P1, P3, P19
	5	P7, P21
Only one student works in Codestrates	8	P8, P14, P15, P22
	5	P4, P9

Table 7: Primary working styles of pairs. NO.: The number of assignments on which the pair used this working style.

Furthermore, the teaching assistants reported that this year assisting students also was easier, as the students just needed to provide a link to their codestrate in order for teaching assistants to help them.

Lower threshold to use

Also, students reported in the questionnaires and the focus group that they see a potential for other use cases in the shareability, for instance, when sharing documents with people with a non-technical background.

> *If you want to quickly send something to someone who is unfamiliar with web development, you can just send a link instead of all files that are associated with the project.*
>
> — P2

Missing permissions

One problem that occurred at the beginning of the course, however, was that students forgot to edit the permissions of newly created codestrates so that their partner could edit the codestrate, too. By default, the permissions were set to only reading access for other users than the user who created the codestrate. This caused confusion, as students did not know why it did not work, but was resolved quickly by informing students on how to set the permissions properly.

Finding 1.4: Submission Repository as a Fail Safe

Multiple pushes per week

Three of the four students in the focus group reported that they would use their submission repository as a way to back up their current state of work. They pushed assignments multiple times per week to their submission repository. This was also confirmed by the log data, which indicates an overall average of 6.3 pushes into the submission repository per week per pair.

> *We mainly used it [pushing our solution to the submission repository] as a checkpoint. Especially when integrating new functions ...*
>
> — F3

Students reported that they pushed their current solution to their submission repository as a "checkpoint" (F3) or fail safe when trying out new code that could possibly destroy their codestrate (e.g., by an endless loop). However, not every pair worked like this; the pair of F1 just pushed the completed assignment once at the end of each week. They always finished the assignment together in one go.

5.3.2 *Web-based Computational Notebooks*

Sections and paragraphs

Computational notebooks differ from classic file-based development environments in their structure. Instead of files and folders, users structure their programs in sections and paragraphs. The differences

to better understand their suitability as a development environment are investigated.

Finding 2.1: "No Setup" Time Assists Initial Phases of Development

Concentration on important things

Both students and teaching assistants reported that they liked how they could directly focus on the content of the assignments, in contrast to setting up the project in a software development environment. Since the computational notebook runs in the web browser, it works across multiple platforms. All students that participated in the focus group reported that the easy setup helped them to focus better on solving the assignments.

> *It takes the work out of one's hands. It lets one concentrate on all of the important things. Implementation etc. How to realize things.*
>
> — F3

They, for example, did not have to find out how to add libraries to their project, e.g., downloading files and adding them to their project. They could add a library through a dialog from within the notebook.

Plug-and-play

Teaching assistants also noted how quickly the process of getting to the actual work was. T2 described it as "plug-and-play" where "one can just open a new codestrate or an existing project and it would directly run, unlike projects in folders that need some kind of integrated development environment." This type of "plug-and-play" behavior also applied to the way of working of packages.

> *What in principle also is another advantage over IDEs is that it is working, so to say, plug-and-play. ... assuming another teaching assistant created a package for the eighth sample solution ... she could just tell me "Hey, pull that." and it would just work.*
>
> — T2

Finding 2.2: Linear Structure Assists Students

Division into sub-problems

The linear structure of computational notebooks supported students in solving the assignments. It encouraged them to focus on one part of the assignment at a time. First writing the body of the application with HTML, then styling it with CSS, and lastly adding interactivity through JavaScript. This also helped them to split up work with their partner. For instance, one student wrote the HTML and JavaScript while the other styled it using CSS.

> *I also like it [the structure] very, very much, because I have this division into sub-problems. From HTML to CSS to JavaScript.*
>
> — F2

Restricted customizability

In contrast, T2 and T3 reported that the linear structure restricted the customizability of their codestrates. Not being able to watch two paragraphs side by side within a codestrate forced them to open the same codestrate in multiple tabs or only working in one paragraph at a time. Similarly, F2 reported that the fixed-size preview paragraph in the assignments hindered developing a responsive interface.

Finding 2.3: Literate Computing Blurs the Separation Between Development and Use of Software

Seamless development

All three teaching assistants reported that they liked the directness Codestrates provided to users when programming with it. T1 reported that the programming felt more direct and seamless in comparison to the programming with a code editor and compared it to last year, where one would write code in a code editor and then open the HTML file in the web browser to preview it. In Codestrates, following the literate computing approach, the workflow felt more seamless as no application needed to be switched and code changes, like CSS, are directly applied, and files do not need to be saved explicitly. T1 reported that, when switching back to a code editor after using Codestrates for a while, he needed to remember saving files and to reload the preview manually in the browser.

> *In hindsight I think it [manually saving files, opening their preview in the browser, and switching applications] is effortful. . . . To a certain extent I already was used to it working automatically.*
>
> — T1

Unfamiliar directness

T3 felt the same about the workflow and furthermore described that this directness is contrary to what one is used to in programming: a clear separation between the development and use of an application. Especially when editing body paragraphs that can either be edited using the HTML editor or by directly writing within the elements using the `contenteditable` attribute — similar to a WYSIWYG (What You See Is What You Get) editor.

> *On the one hand it is awkward because it is unfamiliar, but on the other hand — because it is so immediate — I think it is interesting . . . I have the feeling I directly edit something. Because usually it is like: in the one [application] I edit something and in the other I see it.*
>
> — T3

5.3.3 *Reprogrammability and Extensibility*

Malleable development environments

Being based on Webstrates makes Codestrates an inherently reprogrammable platform, up to a level where its users can modify fundamental functions of a codestrate. Besides, Codestrate Packages allows users to easily extend or reduce the function set of a codestrate by adding or removing packages. The combination of reprogrammability and extensibility provides users with much power and creative freedom, on how they can alter a codestrate to fit it to their needs and preferences. This subsection will investigate how these two properties influence users and their way of working, and — equally important — what drawbacks this kind of freedom can bring with it.

Finding 3.1: Reprogrammability Brings Both Benefits and Barriers

Freedom in use

Teaching assistants mentioned that the creative freedom that the web-based platform offered was a benefit: one could do things like adding GIF (Graphics Interchange Format) files or videos into slides or assignments, transcluding sample solutions directly into slides, or altering the codestrate itself. As the platform is reprogrammable, it also allowed users to extend or modify it. T3 stated that this, on the one hand, is "fascinating" but on the other hand, perhaps not well suited for beginners as one needs "a sort of an understanding of it in order to assess its behavior."

For the students this freedom and reprogrammability sometimes caused problems. For example, when trying to remove entries from a to-do list, some pairs accidentally selected and removed the wrong HTML elements of a codestrate, causing the codestrate to no longer function properly, as system sections were deleted in the process.

Finding 3.2: Mostly Programming and Collaboration Packages Used

Understandable concept

The concept of extensible software was understood by all participants of the focus group and all teaching assistants. Students compared adding packages to adding libraries in other development environments. However, they liked that adding them was easier and faster as adding libraries to a project in an IDE.

> *What was quite good with the packages, is that they were easy to load. I mean things like jQuery or so. Instead of inserting something [into the code] one could just load the package.*
>
> — F4

Used packages

Students mostly used the programming packages *jQuery* and *Materialize* that made the same-named JavaScript libraries and CSS styles available to them. Next up were packages that supported their collaboration: the *Avatars* package that displays users who are currently

working on a codestrate, and the *Shared Pointers* and *Remote Cursors* that allows users to see cursors and text selections of other users.

No easy exploration

Apart from these, participants did not try out other experimental packages that were made available to them, yet, were not relevant to their assignments. Students stated that they were not necessary for them and had no interest in trying them out. Teaching assistants reported that, besides not being directly relevant, packages also were not easy enough to explore. They stated that it was cumbersome, that packages first needed to be installed before one could grasp what exactly the package was doing.

> *I always find it difficult to explore such features without an image. Because it says "Text Tools" but you don't know what you will really get.*
>
> — T2

Finding 3.3: Too Many Repositories for Teaching Assistants

All in one place

Many of the things related to the tutorial of the course were done in Codestrates: the distribution, submission, and correction of assignments, the distribution of sample solutions, the distribution of slide decks of the tutorial, as well as other organizational information like a list of students, their pair partners, and their received scores in the assignments. This was perceived as an advantage by all three teaching assistants compared to the years before, where they had they used multiple platforms like ILIAS, Dropbox, a shared network folder, and emails with attached documents.

> *The whole organizational workflow is handy. I do not have thousands of folders on my hard drive anymore.*
>
> — T2

Lacking overview

However, as many things were done in Codestrates, there were also many repositories for assignments, submissions, slide decks, and corrections. All teaching assistants reported that this forced them to open multiple tabs in their browsers in order to access all of these things and that there were almost too many repositories.

> *On the one hand, it is all in one place instead of being spread across various files but, at the same time, we now have I don't know how many "hundred codestrates" in which we work.*
>
> — T3

Codestrate explorer

The teaching assistants wished to have some better overview of all their codestrates and the contents that they used for the course. They also suggested adding a kind of "explorer" to browse one's codestrates, similar to how they could browse files on a computer.

Finding 3.4: Missing Export Functionalities

Compatiblity to other systems

A downside of using codestrates and literate computing as opposed to a file-based IDE was found to be the (yet) lacking exportability into standalone applications. When being asked whether they could imagine using Codestrates for future projects, some students and teaching assistants answered that they probably would not use it as they would not know how to export their results. For example, two groups wished for the possibility to export individual paragraphs into files and were unsure how their application could be exported to a regular web server.

> *I would like to have the possibility to get the complete HTML and JavaScript files, so I could use them on a web server.*
>
> — P22

Finding 3.5: Hidden Logic Can Cause Confusion

Wrapper around the application

Another drawback that T3 pointed out is that one does not have the full control over the project anymore. As the applications created by participants run within a codestrate and libraries are imported using packages, the JavaScript code written by users is not the only code that is being executed, but also the code of the system sections and the installed packages of a codestrate. This stands in contrast to file-based programming, where, in web development, users can usually specify all files and code that is being executed within their website or web application.

This can cause confusion when users do not know whether a bug is caused by their code or by the system's code, i.e. the code of the system sections and the installed packages. For example, two packages that were used by the students interfered with each other so that sometimes the initialization of elements failed although the code of the students was correct.

5.4 DISCUSSION

Discussions by topic

In the following subsections, the findings of this study will be discussed. Like the findings, the discussion is split into the same three topics. Each subsection will discuss the findings of its respective topic.

5.4.1 *Collaborative Working Styles*

Synchronous and asynchronous collaboration

The findings show that pairs adopted various working styles to solve the assignments. While many pairs collaborated synchronously, asynchronous collaboration was in fact the most common working style among pairs. This indicates that synchronous (cf. Yim et al. (2017))

and asynchronous collaboration both benefit from real-time synchronization and easy shareability as offered in Codestrates. The asynchronous collaboration style is similar to the results of Suwantarathip and Wichadee, where students could help each other on writing assignments at any time, allowing collaboration "without restriction of time and space" (Suwantarathip and Wichadee, 2014). Additionally, students used Codestrates not only to solve the assignments but also for coordinating their collaboration — similar to results of Olson et al. (2017) —, and handover of different states of their submission either via a shared codestrate or the submission repository. It seems that there are similarities between collaborative writing and collaborative programming; future work could investigate how the benefits of collaborative writing platforms could be transferred to collaborative programming environments.

Collaboration capabilities do have some room for improvement nonetheless: students missed tools to keep better track of their partners' work when working asynchronously; technically, these changes are logged in the form of operations by the Webstrates server; however, they are not readily accessible to users. The students also encountered the problem of jumping content when they were working synchronously on the same codestrate.

Trade-off of synchronization

The findings further indicate that there is a trade-off in synchronizing the content of a codestrate when two users use it simultaneously: on the one hand, the synchronization of content at a keystroke-level can hinder individual work, e.g., when contents in a codestrate jump; on the other hand, it is difficult for users to coordinate and maintain awareness when there is not enough contextual information about one's partner's actions. Features like remote cursors or pointers are essential to maintain this awareness — even the mere indication of an avatar that displays whether a user is currently online in a codestrate or not can improve this awareness.

Missing awareness for asynchronous collaboration

While the contextual information discussed above is useful for synchronous collaboration, asynchronous collaboration — the most frequent collaboration pattern — is often overlooked in terms of providing similar types of tools. Future systems should focus on better support of asynchronous collaboration. Being able to keep track of the changes that other users perform could ease splitting up work and putting results together. Such collaborative tools can already be found in platforms of other domains, such as ShareLaTeX or Google Docs. Features like commenting, an edit history, and the highlighting of tracked changes could further support asynchronous collaboration in programming.

Different collaboration modes

In order to address the issue of jumping content when working synchronously, future systems could provide different modes for different working styles. For instance, when dividing and conquering work, one mode could turn off synchronization for the user interface

or allow pinning of paragraphs. When working on the same parts of an application, a more tightly-coupled mode could allow both users to see the cursors and pointers of their partners to create an awareness of what the other is working on. This future work could also be informed by previous research on collaborative sensemaking on interactive tabletops (Isenberg et al., 2012; Tang et al., 2006).

Device-agnostic shareability

Another important aspect regarding the ease of use of collaboration within Codestrates was the sharing via links. It allowed users — both teaching assistants and students — to share documents with one another easily. Being able to open documents on almost all types of devices, including desktop computers, notebooks, tablets, or even smartphones as well as different operating systems, such as Windows, iOS, macOS, Android, or Linux, made it possible for users to collaborate using different device collections as each other. Again, this is a benefit shared with collaborative word processors like Google Docs. This is in line with the work of Tchernavskij et al. (2017), which reports that "research on hypermedia has, since the early days, emphasized collaborative work, distributed access and changing activities."

5.4.2 *Web-based Computational Notebooks*

Quickstart into programming

The lower threshold to set up and use a codestrate allowed users — both students and teaching assistants — to quickly get started and tinker with code. They found themselves comfortable in using sections and paragraphs instead of files; some even compared sections and paragraphs with files and folders, which are arranged linearly. The linear structure of a codestrate supported the students' workflow, as the visual separation of individual paragraphs guided them to split up the assignment into subtasks. This is similar to how computational notebooks for data analysis, like Jupyter (Kluyver et al., 2016), split the analysis of complex data into multiple cells, each of which can be executed on their own.

A more flexible structure

Although Codestrates supports novice users in initial phases, the linear structure of codestrates can also be a limitation for more experienced users. For some tasks, such as the comparison of different paragraphs, this resulted in additional scrolling activities that hampered the ability to compare paragraphs. It would be an improvement for users to be able to break up the linear structure, e.g., showing two paragraphs side-by-side.

Blurring the line

The results of the study highlight the benefits of Codestrates' direct approach compared to file-based development, i.e. the ability to see the results of computations immediately — especially in contrast to the more common separation of development and use. Codestrates' direct approach was perceived positively by participants despite the fact that many of them were unfamiliar with it. They particularly commended that in Codestrates the results of one's implementations

are directly visible, which reduces the number of steps needed when iteratively developing and testing an application. Interestingly, this directness is not novel in all domains, for instance, word processors have been offering WYSIWYG editors to modify documents for a long time — many users, though, do not even know that other types of editors exist. In programming, however, developers are accustomed to the necessity to compile code or reload a web page in order to apply their changes.

5.4.3 *Reprogrammability and Extensibility*

Two sides of the coin

The *freedom* that the reprogrammability of a platform like Codestrates provides is a double-edged sword: on the one hand, it offers users many new possibilities to use and develop software; on the other hand, however, these new possibilities are mostly unfamiliar to users who are accustomed to a clear separation between development and use in file-based development environments. When used the wrong way, this freedom can be more of a burden for users — no matter how extensive the possibilities are.

This is a problem, especially for novices of such systems, as more experienced users such as — in the case of this study — web developers "know their tools", and therefore have a better foundation to understand the system and to evaluate its capabilities. This understanding also makes it easier for them to avoid accidentally reprogramming functions of the system and thereby possibly making it inoperative. A possible solution would be to provide a sort of *beginner mode*, which, when activated, prevents users from altering the implementation of the low-level functionalities of a codestrate — whether intentionally or unintentionally.

Better presentation for packages

Although participants understood the concept of an extensible system, the lack of possibilities to explore available packages was hindering to the exploration process of new packages. These findings indicate that the presentation of packages or any type of extension of functionality needs to be properly presented to users. It should be clear — right away — what kind of functionality a package offers and which benefits it holds for the user. Inspiration can be drawn from other platforms that use the concept of an extension of functionality, for instance, app stores of mobile operating systems, or stores for browser extensions. By providing images, a more extensive description, and possibly videos or GIFs to illustrate the functionality, the exploration of packages could be made easier.

Versatile platform

The results of the study show that computational notebook systems like Codestrates do not only have the ability to act as a development environment for developers, but they also have the potential to act as an entire learning management platform for a course. Its package management, permissions system, and reprogrammability make it a

versatile platform whose potential can be exploited in unexpected ways. This indicates that computational notebooks have the potential to be a suitable platform for even more use cases other than data analysis and software development. However, there are optimizations needed in order to comply with the requirements of such systems as, for example, the high number of repositories already became a minor problem for the teaching assistants in this study.

Missing interfaces

Although Codestrates allowed exporting some content, such as slide decks as PDF files, it lacked the ability to export content in more sophisticated ways. It is, for example, not yet possible to export one's implemented application or web page into a collection of files that allow for the deployment on a regular web server. For instance, other platforms like Jupyter allow exporting notebooks as HTML or PDF files; however, they do lose their interactivity in the process. This (yet) missing interface to other platforms or applications could hinder users from using platforms like Codestrates, as the results that are created on this platform are restricted to be used on that very platform.

6 IMPLICATIONS AND FUTURE WORK

The findings and obstacles uncovered within this research are used to distill implications for the design of future systems. The first section will cover various ideas and concepts for future prototypes. Next, the second section will report on the limitations of the conducted study, and lessons learned in the process. Further, it will provide ideas and suggestions for future research and studies on this topic.

6.1 IMPLICATIONS FOR DESIGN

This section will cover various implications for the design of future systems and prototypes. The implications and ideas are derived from the findings of the study as well as the suggestions of participants in the questionnaires, interviews, and focus groups. The implications are grouped into four groups.

6.1.1 *General*

USABILITY IMPROVEMENTS: Codestrates provides a large enough feature set and extensibility to be successfully used in a study in the wild, however, being a research prototype, it does not provide the same level of features or usability as commercial IDEs or systems do. In order make its use more intuitive and self-explanatory, Codestrates needs to catch up on these. Minor improvements such as better feedback, e.g., whether the pushing of a package is finished or still running, or better-explained buttons, e.g., by adding labels when hovering over them, could improve usability and lower the threshold for new users at a low cost for implementation. A better documentation, how-to guides, and an FAQ (Frequently Asked Questions) page would also flatten the steep learning curve for new users.

TEMPLATES FOR USE CASES: The use cases of users vary from user to user. When creating a new codestrate, users first have to install the packages they need before they can start with the actual task itself. Templates of codestrates for specific use cases, e.g., writing, developing, or drawing, could reduce this overhead of always having to download the necessary packages beforehand. Although this is technically possible by creating a codestrate with the packages, tagging it, and afterward copying the prepared codestrate with the installed packages instead of copying a new empty codestrate, it is not easy to come up with.

A possible solution would be to create a codestrate, that allows users to create and edit templates of codestrates for specific use cases. Such a *template manager* could be the starting point for users to create new codestrates, allowing them easy access to their templates. The inspiration for such a manager could be drawn from applications like Microsoft Word, which offers a variety of templates when creating new documents (see Figure 23).

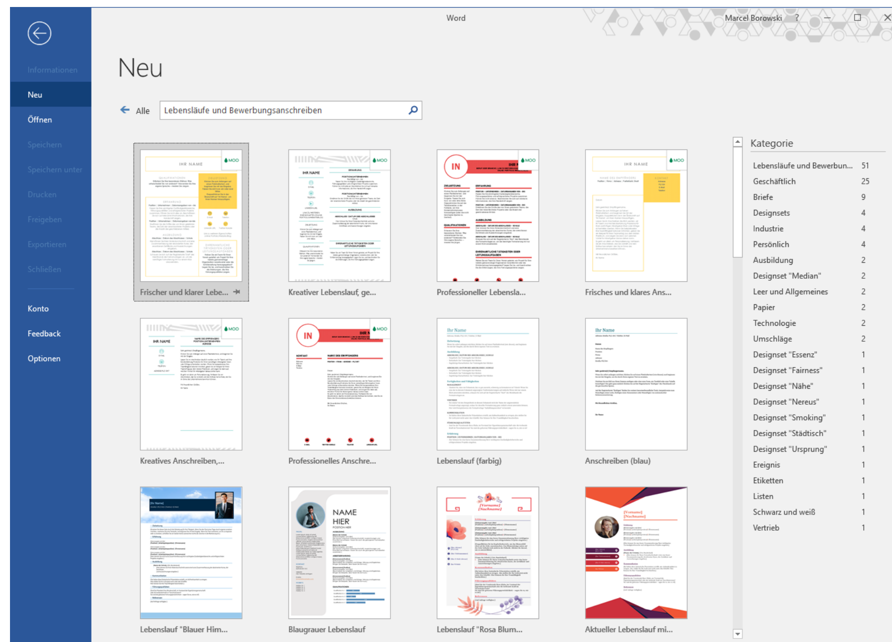


Figure 23: The template gallery in Microsoft Word.

WIZARD FOR NEW CODESTRATES: One suggestion of the teaching assistants was to create a wizard for the creation of new codestrates. The idea is tightly coupled with the previous implication of having templates of codestrates for different use cases. When creating a new codestrate, a wizard could guide users through questions in order to identify the use case of a codestrate. It could then offer users the possibility to install the packages that are recommended for that kind of use case.

CODESTRATE EXPLORER: Participants reported that when using many codestrates or repositories the overview over one's documents is suffering. Being used to files and folders, participants expected to be able to see all their documents, i.e. codestrates, in an overview. Technically this is possible, for example, by creating bookmarks in the web browser or storing the links as desktop shortcuts on the device. However, this would defeat the benefit of the platform being independent of local files or bookmarks. Furthermore do codestrates, that were created by users, not automatically appear in the bookmarks of a web browser — users have to create the bookmarks manually. This

is contrary to how files work, which are automatically visible in the file system when a user creates them. A solution would be to automatically track the codestrates users create and store them in a kind of *recently used codestrates* list within Codestrates. Additional functionality such as grouping or tagging codestrates could make managing higher numbers of codestrates easier and keep the platform independent from local files.

EASIER CUSTOMIZATION OF THE USER INTERFACE: Although the linear structure of a codestrate was useful for the process of splitting tasks into sub-tasks, it also proved to be limiting for some users. Future designs should consider the possibility to make customizations of the user interface possible and easy to use. One idea is to show content side by side, for instance, the HTML editor could be displayed beside the preview of the content it is editing, as opposed to the current version which limits the location of the HTML editor to the bottom of the screen. Another feature that teaching assistants missed, was a way to customize keyboard shortcuts in a codestrate easily.

Another problem regarding the linear structure of computational notebooks was the frequent scrolling when switching back and forth between two paragraphs. This could be solved by allowing users to view paragraphs in tabs—just like in IDEs or web browsers. This way switching between them would be more comfortable. As this could interfere with close collaboration, different modes for collaboration could solve this issue (see *Loose and closely coupled modes* below).

6.1.2 *Collaboration*

LOOSE AND CLOSELY COUPLED MODES: To address jumping content and distraction by awareness tools, such as shared pointers, when working synchronously, future versions of Codestrates or computational notebooks could provide different modes for different styles of working. For instance, when splitting up work in a divide-and-conquer fashion, one mode could decouple the interfaces of users or allow them to pin paragraphs, so edits of their partner in other paragraphs do not interfere with their local interface. In this mode, the customization capabilities for each user could be more extensive, as customized changes would not be mirrored on their partner's interface. When working on the same parts of an application, a more coupled mode could allow both users to see cursors and mouse pointer of other users and thereby creating an awareness of what the other user works on.

USER MANAGER AND FRIENDS LIST: When students created new codestrates, they first had to grant their pair partners writing permission for these codestrates. This was done by manually entering

their GitHub username. To ease this process, a user manager could be added that autocompletes usernames and allows users to search for other users. A *friends list* could make finding users one frequently collaborates with faster. Additionally, it could be made configurable to grant one's friends write permissions in newly created codestrates automatically.

CHANGE HISTORY: As discussed in the previous chapter, asynchronous collaboration — being the most frequently used style of collaboration — needs more tools to aid it. One suggestion by students was to have a change history to follow up on the changes made by a partner since the last visit of the document. This could be realized by creating a list displaying the changes of all users — similar to Google Docs or ShareLaTeX. Changes thereby could also be highlighted within the code, again, ShareLaTeX implemented such a feature in their online LaTeX editor (see Figure 24).

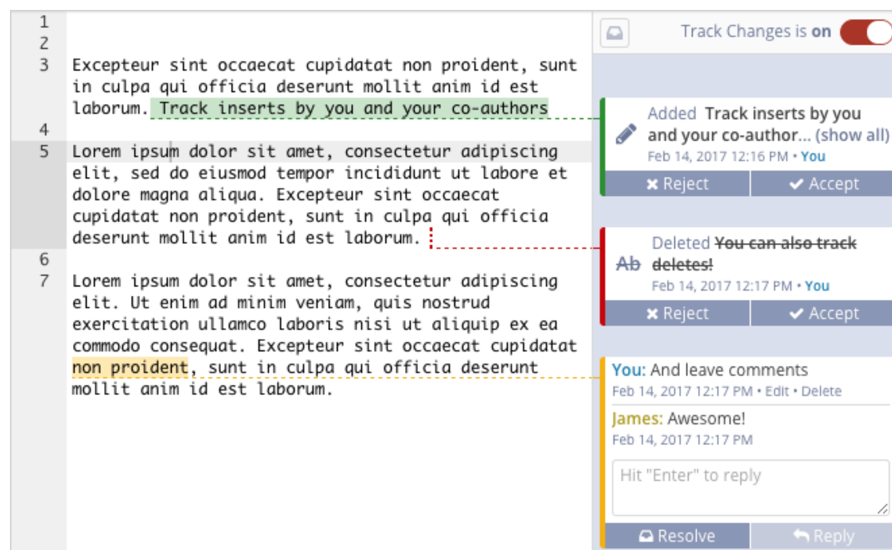


Figure 24: The track changes feature of ShareLaTeX. (ShareLaTeX Blog, 2017)

6.1.3 *Robustness*

SAFE MODE: In some instances, participants of the study broke their codestrates so they would no longer function. As mentioned earlier, there are different use cases, and in some of them, users do not need and do not even want to alter the functions of the system itself. For example, when just using the system to write or create a slideshow presentation users want to use the system without the fear of breaking it and losing their work as a consequence. To prevent this from happening, Codestrates or other future systems that support the reprogrammability of the whole system should support a *safe mode*, that prevents users from accidentally breaking system func-

tions when activated. Only when disabled, the functionality of the system should be modifiable by users.

EASIER RESTORING: In case of breaking a codestrate or deleting one's content by accident, the recovery process should be made easier. At the moment, users need to manually examine older versions of ones codestrate and then restore that version by using the HTTP API of Webstrates. Future tools could make this process easier by providing a user interface or dashboard for the recovery process of older versions, and the browsing of the history of a codestrate (similar to the *Change history* discussed above). Inspiration can also be drawn from other research, for instance, using an automated version control that stores "more program-rich metadata" to allow "a variety of context-specific version searches" as discussed by Kery et al. (2018).

6.1.4 *Package Management*

PACKAGE VERSION HISTORY: Currently, packages can be pulled from either the current version of a repository, one of its tags, or a specific version number. Technically, this allows pulling older versions of a package from a repository — given an older version of the package was pushed to the repository before. However, this is not very convenient, as versions and tags need to be sought manually. Future systems could provide a history for each package — possibly even storing that history within the package itself in order to make the version history exportable. This would benefit users who want to retrieve an older, possibly overwritten, version of their package, or who want to use an older version of a package because the current version is conflicted with another package.

PROPERTIES EDITOR: Some participants of the study found it inconvenient editing the properties of their packages as plain JSON code. It is prone to errors and typos and does not provide any feedback to the user, whether his input is correct. Instead, a user interface to edit the properties could be added, which allows to make edits to the properties easier and ensure the structural validity of the JSON code. Such an editor could be realized using existing libraries such as JSON Editor[1], which would allow editing the properties using an input form.

UPLOAD MANAGER FOR ASSETS: Students and teaching assistants wished for an upload manager for assets, i.e. images or other attached files. While uploading assets via drag-drop caused only little problems, managing the assets later was more of a problem. Students sometimes were not sure which assets were already uploaded and

1 JSON Editor: http://jeremydorn.com/json-editor/ (accessed November 15, 2018)

what the exact file names were. Some students, for example, entered the wrong file names in the assets section of the properties of their assignments and later, when pulling their submission into another codestrate, wondered why their images were not working.

An asset manager within every codestrate could resolve this problem by providing an overview of all assets that are attached to the codestrate and their, if available, previous versions. This manager could be incorporated into the properties editor discussed above, allowing users to easily assign assets to packages without having to tinker around with entering exact file names.

BETTER PRESENTATION OF PACKAGES: To improve exploring and browsing of packages, future work should deal with a better presentation of packages. Inspiration can be drawn from app stores or plugin galleries. WordPress[2], for instance, presents plugins in a more elaborate form than Codestrate Packages does (see Figure 25). These plugin or package pages could also contain examples or a link to a codestrate, showcasing the package.

6.2 LIMITATIONS AND FUTURE WORK

Novice users

The study investigated the use of Codestrates for the development of interactive systems. This was evaluated in the context of teaching undergrad computer science students user interface design patterns. Their experiences and working practices might significantly differ from other demographics, for example, senior software engineers. As a consequence, the participants might have had a lower barrier to adopting computational notebooks as a development environment due to them being less entrenched in traditional software development practices. Additionally—as the participants were still learning to program—their workload was threefold: learning design patterns, learning how to develop applications with HTML, CSS, and JavaScript, and adapting the development environment.

Restrictions by assignment structure

In order to comply with this additional workload, the structure and extent of each assignment were predefined, in order to facilitate getting going without overwhelming students with large projects. One of the advantages of using computational notebooks like Codestrates as sandbox environments is that they can support both students solving the assignments as well as the teaching assistants correcting them. Yet, allowing users to leverage the full potential of the Codestrates platform (e.g., to develop prototypes for multi-user scenarios) in combination with a single project that spans the entire semester might lead to different collaboration patterns.

Limits of log data

The analysis of the log data revealed students' working styles in which the majority of students worked mostly at different places and

2 WordPress: https://wordpress.org/ (accessed November 15, 2018)

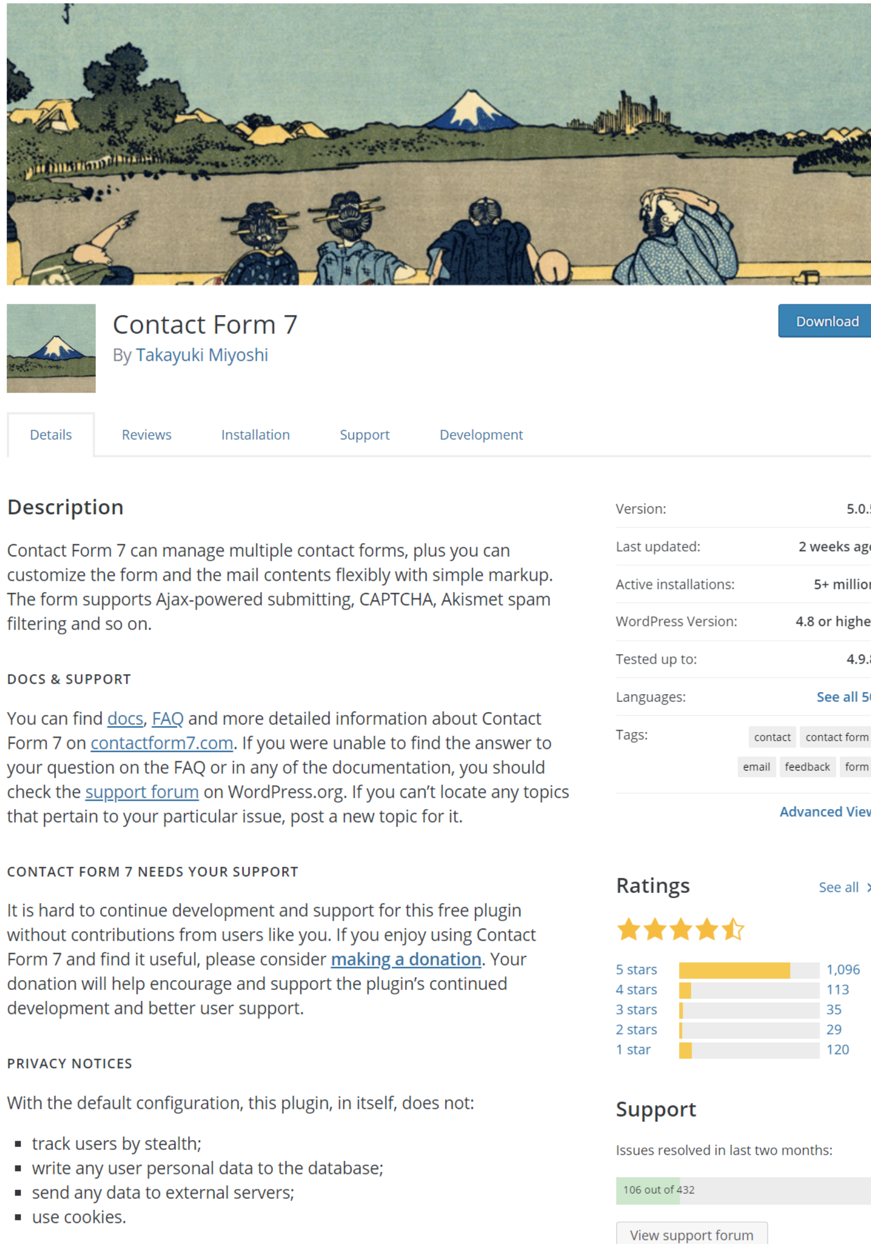


Figure 25: A plugin page of WordPress.

at different times. Codestrates allowed them to combine or merge their individual solutions by, for instance, visiting the same codestrate, using their submission repository, or by copying and sending parts of their code to their partner. However, the log data does not provide insights into interpersonal communication, such as face-to-face or via messaging apps, that was used additionally to coordinate activities. The log data showed several instances where only one student was actively working on the assignment in a notebook. This does not necessarily mean that these pairs did not collaborate; as the questionnaires showed, some pairs worked at the same time and place during these assignments, e.g., sitting together in front of one laptop. Additionally, other possibilities, like one student working on a local code editor or code playground while the other one combines their solution in a codestrate, cannot be excluded. Thus, these aspects of collaboration are not reflected in the log data.

Future research

Future work needs to be conducted to continue research in this direction. This will include studies with more experienced developers and projects with a longer duration and larger scope. Studies could also be conducted in the form of a hackathon, in which participants implement applications over the duration of a few days, allowing for in-situ observation instead of self-reported feedback and log data — overcoming the limitations of this study.

7 CONCLUSION

Document-centric future

Motivated by the idea of a future where computation is no longer siloed in applications but part of documents, and therefore easily shareable and malleable, this work investigated the effects of the Codestrates platform and its package management on collaborative programming. In particular, this work focused firstly on collaboration in a computational notebook, secondly on how the structure of computational notebooks influences programming, and thirdly on how the malleability and extensibility of packages affect programming.

Activities and blurred lines

The concepts of the human activity model and instrumental interaction illustrated the importance of tools that fit the users' task at hand. Following the design principles reification, polymorphism, and reuse, instruments can be mixed and matched for the respective task. While instrumental interaction blurred the line between applications and documents, literate computing showed how computational notebooks blur the line between the development and use of an application. Reviewing work on computational notebooks, code playgrounds, and online office suites revealed the benefits and shortcomings of these systems and helped to understand their use. While computational notebooks are mostly used for exploration purposes by data scientists, their use for programming applications, and web-based collaboration is under-explored. Research focusing on online office suites, however, proved that such web-based platforms are beneficial for collaborative writing tasks, as they support users and encourage them to help each other.

Codestrate Packages

Inspired by the concept of instrumental interaction and following the document-centric model, Codestrate Packages allows for documents, i.e. codestrates, that contain the functionality, i.e. packages, themselves, and therefore no longer require separate applications in order to be viewed or modified. Building on top of Codestrates and Webstrates, this allows for the packages and documents to be malleable, shareable, and distributable.

Study "in the wild"

To investigate the influence of such a system on its users, their way of working, and the collaboration among users, a 13-week long study was conducted in a university course. The malleability of Codestrate Packages allowed it to make slight modifications to the system in order to use it in the course. Both qualitative and quantitative data from questionnaires, interviews, focus groups, and data logging was collected throughout the study and subsequently analyzed.

The results of the study provided insights into the distinct collaboration patterns of student pairs when implementing small interac-

tive applications. The findings highlight the importance of a computational notebook that can support multiple working styles, as users incorporate diverse working styles ranging from synchronous, asynchronous, to no collaboration at all, as well as from co-located to remote settings. The structure of computational notebooks made it easier for beginners to get started, however, sometimes restricted more experienced users because of the linearity of notebooks. The reprogrammability proved to be a double-edged sword, as it allows users to modify and customize codestrates and packages significantly, yet carries the risk of novices accidentally breaking the system. This reprogrammability and extensibility make Codestrates a versatile platform that can be used for the distribution and submission of assignments. Even though, for some use cases, the platform still misses some interfaces and export options to be used in more projects.

Implications

Based on the findings, implications for future systems and further development on Codestrates were provided, including general usability improvements, tools to better support asynchronous collaboration, and more robust code execution to prevent the accidental breaking of documents. Limitations of the study gave directions for future work. Future studies could, for instance, investigate how collaboration unfolds among more experienced users, or observe users during a study in the form of a hackathon to overcome the limits of the log data.

Outlook

This work demonstrated how complex and nuanced the task of collaborative programming is. The interplay between shareable dynamic media in the form of Webstrates, literate computing in the form of Codestrates, and the document-centric model in the form of Codestrates Packages proved to be well-suited for the execution of this task — even in an unrestricted study "in the wild."

While still having some technical weak points which need to be strengthened, this research shows huge potential. With applications and devices becoming more diverse every year, Codestrate Packages offers a platform that transcends device bounds and is malleable down to the lowest system functions, thus providing a solid and future-proof platform to build upon. Furthermore, the operational log provides a powerful tool for tracking users' edits during studies. It will be interesting to see in what ways future research will make use of this platform. Badam et al. (2018), for instance, used Codestrate Packages to create Vistrates, "a literate computing platform for developing, assembling, and sharing visualization components." Besides continuing to investigate collaborative programming, e.g., among more experienced users, other application areas of Codestrate Packages could include cross-device interaction, and thereby slowly pave the way to Weiser's original vision of ubiquitous computing.

CONTENT OF THE FLASH DRIVE

The flash drive that is attached to this thesis contains the supplementary material:

- The thesis in digital format.
- A video showing the workflow of processing a computer science assignment.
- The Tableau project of the log data evaluation.
- A link to codestrate, containing all assignments.
- A downloaded archive of the codestrate, containing all assignments.

B

DEMOGRAPHIC QUESTIONNAIRE

The following subfigures show the demographic questionnaire as realized with Google Forms. The questionnaire was translated into English for this appendix; students received it in German. Each subfigure shows one topic of the questionnaire.

Personal Data

Age

My answer

Gender

- Male
- Female
- Other:

Study Program

- Bachelor's
- Master's
- Other:

Major

- Computer Science
- Information Engineering
- Other:

Semester
Please enter your current semester. As a freshman you should enter „1".

My answer

Specialized Course *
Are you participating in the course as a specialized (Computer Science or Information Engineering) or non-specialized course (other majors)?

- Specialized
- Non-specialized

(a) Personal Data.

Figure 26: Questions of the demographic questionnaire.

Programming Experience

How would you rate your programming experience?

	1	2	3	4	5	
Beginner	○	○	○	○	○	Expert

Which web technologies (z.B. HTML, CSS, JavaScript) and which version of them (HTML4, HTML5, ES2015, ES2016, ES2017, ...) have you already used for development?

My answer

Which programming languages have you used before?

- ☐ Java
- ☐ C#
- ☐ C++
- ☐ C
- ☐ Haskell
- ☐ Prolog
- ☐ JavaScript
- ☐ Python
- ☐ Other:

Do you also program in your free time or just as part of your study program?

- ○ Also in my free time
- ○ Only as part of my study program

(b) Programming Experience.

Project Experience

In how many projects have you participated so far?

My answer

Overall, what was the major way of working in these projects?

- ○ Remote at the same time
- ○ Remote at different times
- ○ Co-located at the same time

(c) Project Experience.

Figure 26: Questions of the demographic questionnaire.

Computer Use

How would you rate your computer knowledge?

	1	2	3	4	5	
Beginner	○	○	○	○	○	Expert

How many hours per day are you using a web browser?

- ○ Less than 1 hour
- ○ Between 1 and 2 hours
- ○ Between 2 and 4 hours
- ○ More than 4 hours

Which devices are you using for using a web browser?

- ☐ Computer
- ☐ Smartphone
- ☐ Tablet
- ☐ Other:

Have you ever used browser-extensions in your web browser?

- ○ Yes, more than 3
- ○ Yes, 1 to 3
- ○ No

Which browser-extensions are you using and for what task?

My answer

(d) Computer Use.

Collaborative Web Platforms

Which of the following platforms have you used already?

- ☐ Google Docs, Spreadsheets or Presentations
- ☐ Office Word, Excel or PowerPoint Online
- ☐ Dropbox Paper
- ☐ ShareLaTeX
- ☐ Overleaf
- ☐ Other:

If you have used platforms like Google Docs or Office Online so far, for what tasks did you use it?

My answer

(e) Collaborative Web Platforms.

Figure 26: Questions of the demographic questionnaire.

C INTERVIEW AND FOCUS GROUP QUESTIONS

In the following the questions and topics of the interviews and focus groups throughout the study are listed.

C.1 WEEKLY INTERVIEWS

The following questions were asked to the teaching assistants in their weekly interviews:

Way of Working

- How confident do you feel in using the package management?
- Did the package management influence your way of working (e.g., in contrast to other development environments)? If so, in what way?
- What percental amount of the workload of the assignment did the package management took up?

Correction

- Did you correct assignments?
- Did you correct assignments alone or in a group?
- Did any students reach out to you because of problems with Codestrate Packages? If so, what problems for instance?

General

- Did you have problems in using Codestrates or the package management?
- Do you have suggestions for improvement or wishes for Codestrates or the package management?

C.2 MID-TERM FOCUS GROUP

The following questions were asked to the teaching assistants in their mid-term focus group:

General

- How satisfied are you with the system so far?
- How was your learning process?

Codestrates

- Is the structure of a codestrate understandable?

Packages

- Is the concept of modular software understood?
- Did you try out other experimental packages of the Codestrate Packages repository?

Differences to other platforms

- Differences to other coding platforms like JSFiddle or CodePen?

Own Development

- Have you developed something on your own, unrelated to the course or assignments?
- Which were obstacles or problems why you didn't implement something on your own?

Future Usage

- What kind of future projects could be developed using Codestrates?

C.3 END-TERM FOCUS GROUP (TEACHING ASSISTANTS)

The following questions were asked to the teaching assistants in their end-term focus group:

Literate Computing

- Have you used a "computational notebook" before?
- Is the "notebook-like" structure suited for Codestrates and programming?

Differences to IDEs

- What differences do you see between Codestrates and other IDEs? Pros and cons? Also regarding the way of working?
- What was missing in Codestrates compared to an IDE?
- Looking in the other direction, what does Codestrates do better than other IDEs?

Teaching

- How did you like Codestrates for the handing out and handing in of the assignments and slides? Also compared to ILIAS?

Conclusion

- So after all, what is your tendency? Do you think one could use Codestrates again for exercises? As a conclusion over this semester?

C.4 END-TERM FOCUS GROUP (STUDENTS)

The following questions were asked to the computer science students in their end-term focus group:

Codestrates

- Was the structure understandable? Sections and paragraphs?
- How did you like the linear structure of a codestrate? Also compared to a code editor with tabs and so on?

Packages

- Was the concept of extensible modular software understandable? That one can add functions to a software? Do you maybe know it from somewhere else?
- Do you know other systems that work similarly? Like having packages?
- Apart from jQuery and Materialize, did you use other packages? For example the canvas or collaboration packages?
- Have you looked into the packages the "Codestrate Packages" repo? Have you tried out the packages there? Or have you just focused on the assignments?
- Would you have ideas what kind of packages were missing? Maybe also for collaboration?

Way of Working

- What software did you use so far for programming? What programming languages?
- How was the learning curve?
- How did you collaborate with your team partner? How did you manage who does what?
- When you then worked together, have there been any problems?

Pros and Cons

- Compared to an IDE, what were the biggest benefits and drawbacks of Codestrates? Also thinking about collaboration?

Own and Future Development

- Have you experimented with Codestrates apart from the assignments? Or did you implement an own application?
- Could you image using Codestrates for future projects?
- So what is you conclusion to the system? More positive or negative?

C.5 END-TERM INTERVIEW

The following questions were asked to the non-computer science student in the end-term interview:

Codestrates

- Was the structure understandable? Sections and paragraphs?
- How did you like the linear structure of a codestrate? Also compared to a code editor with tabs and so on?
- How did you like the user interface overall? Was it too technical? Did you find your way around in the beginning?

Packages

- Was the concept of extensible modular software understandable? That one can add functions to a software? Do you maybe know it from somewhere else?
- Packages you used were mainly the text tools or also others? Like the theme?
- Have you missed any other packages?

Way of Working

- What kind of assignments did you have to solve before this course? Or essays?
- How was the learning curve? Also with the first tutorial assignment in the beginning?

Pros and Cons

- Compared to Word, what were the biggest benefits and drawbacks of Codestrates? Also thinking about working on assignments?

Own Development

- Did you develop something on your own or look into the computer science assignments?

Future Use

- Where could you image using such a system?
- Did you like it for solving assignments?
- Could you think of using it for the assignments of your subject as well?

D

LOG DATA EXAMPLES

This sections will provide examples of the log data collected during the study.

D.1 PACKAGE MANAGEMENT LOG DATA EXAMPLE

The following listing contains one entry of log data of a package installation. The `packageStati` were shortened, to only show the package properties of one exemplary package, it usually contains the properties of all packages of the codestrate and the selected repository.


```
{
    "data" : {
        "userId" : "maski89:github",
        "clients" : [
            {
                "color" : "#3b8d4d",
                "id" : "HkUzIc8sM",
                "clients" : [
                    "HkUzIc8sM"
                ],
                "avatarUrl" : "https://avatars3
                    _githubusercontent_com/u/6458785?v=4",
                "userUrl" : "https://github_com/maski89",
                "displayName" : "Marcel",
                "provider" : "github",
                "username" : "maski89",
                "userId" : "maski89:github",
                "permissions" : "rw"
            }
        ],
        "webstrate" : "/interaktive-systeme-loesungen-private/",
        "packagesToInstall" : [
            "tepFHbfN"
        ],
        "selectedPackageIds" : [
            "tepFHbfN"
        ],
        "packageStati" : [
            {
                "versionChanged" : false,
                "both" : true,
                "localProperties" : {
                    "uninstalled" : false,
                    "id" : "tepFHbfN",
```

```
34                     "name" : "Interactive Systems Setup",
35                     "changelog" : {
36                         "1_0" : "Initial version_"
37                     },
38                     "description" : "Your short description_",
39                     "version" : "1_0"
40                 },
41                 "local" : true,
42                 "icon" : "assistant",
43                 "name" : "Interactive Systems Setup",
44                 "id" : "tepFHbfN",
45                 "prototypeProperties" : {
46                     "uninstalled" : false,
47                     "id" : "tepFHbfN",
48                     "name" : "Interactive Systems Setup",
49                     "changelog" : {
50                         "1_0" : "Initial version_"
51                     },
52                     "dependencies" : [ ],
53                     "assets" : [ ],
54                     "tags" : [
55                         "development"
56                     ],
57                     "description" : "General setup for
                               Interactive Systems assignments_",
58                     "icon" : "assistant",
59                     "version" : "1_0"
60                 },
61                 "prototype" : true
62             }
63         ],
64         "repository" : "/interaktive-systeme-packages/",
65         "action" : "install"
66     },
67     "timestamp" : "2018-04-07T19:06:39_625Z",
68     "__v" : 0
69 }
```


Listing: Example of an log entry of the package management log.

D.2 WEBSTRATES LOG DATA EXAMPLE

The following listing contains one operation recorded by the Webstrates log. The operation was the entering of the letter *a* to a body paragraph.


```
{
    "src": "e7bae891e88f75991c4b7858a804855d",
    "seq": 4,
    "v": 771,
    "op": [
        {
            "li": "a",
            "p": [ 3, 3, 27, 2, 2, 2 ]
        }
    ],
    "m": {
        "ts": 1540973968202
    },
    "session": {
        "_id": "5bd9658034033d49d7ee28de",
        "sessionId": "e7bae891e88f75991c4b7858a804855d",
        "userId": "maski89:github",
        "connectTime": 1540973952560,
        "remoteAddress": "134.34.231.108"
    }
}
```


Listing: Example of an log entry of the Webstrates log.

E

ASSIGNMENT ATTRIBUTES

This sections will provide a table with the results of the coding for each of the six attributes.

	A1	A2	A3	A4	A5	A6	A7	A8
P1	U3	U3	U3	U3	U3	U3	U1	U3
P2	U3	U3	U3	U3	U3	U3	U3	U3
P3	U3	U3	U3	U3	U3	U3	U3	U3
P4	U1	U3	U3	U3	U1	U1	U1	U1
P5	U3	U3	U3	U3	U3	U3	U3	U3
P6	U3	U3	U3	U3	U3	U3	U3	U3
P7	U2	U1	U2	U3	U3	U3	U3	U3
P8	U1	U1	U1	U1	U1	U1	U1	U1
P9	U3	U2	U2	U2	U3	U3	U2	U2
P10	U3	U3	U3	U3	U3	U3	U3	U3
P11	U3	U3	U3	U2	U3	U3	U3	U3
P12	U3	U3	U3	U3	U3	U3	U3	U3
P13	U3	U3	U3	U3	U3	U3	U3	U3
P14	U1	U1	U1	U1	U1	U1	U1	U1
P15	U1	U1	U1	U1	U1	U1	U1	U1
P16	U3	U2	U3	U3	U3	U3	U2	U3
P17	U3	U3	U1	U3	U3	U3	U3	U3
P18	U1	U3	U3	U3	U3	U3	U3	U3
P19	U3	U3	U2	U3	U3	U3	U3	U3
P20	U3	U3	U3	U3	U3	U3	U3	U3
P21	U3	U3	U1	U3	U1	U3	U1	U3
P22	U2	U2	U2	U2	U2	U2	U2	U2
P23	U1	U2	U1	U2	U1	U3	U3	U1

Table 8: Coding of the attribute *users*. U1: Only S1 worked in Codestrates; U2: Only S2 worked in Codestrates; U3: Both students worked in Codestrates.

	A1	A2	A3	A4	A5	A6	A7	A8
P1	T2	T1	T2	T2	T2	T2	T2	T2
P2	T2	T2	T2	T2	T2	T2	T2	T2
P3	T2	T1	T2	T2	T1	T2	T2	T2
P4	T2	T2	T2	T2	T2	T2	T2	T2
P5	T2	T2	T2	T2	T2	T2	T2	T2
P6	T2	T1	T1	T1	T1	T2	T1	T1
P7	T1	T1	T1	T2	T2	T2	T2	T2
P8	T2	T2	T2	T2	T2	T2	T2	T2
P9	T2	T1	T2	T2	T2	T2	T2	T2
P10	T2	T1	T2	T1	T1	T1	T1	T1
P11	T2	T2	T2	T2	T2	T2	T2	T2
P12	T2	T2	T2	T2	T2	T2	T2	T2
P13	T1	T1	T1	T1	T1	T1	T1	T1
P14	T2	T2	T2	T2	T2	T2	T2	T2
P15	T2	T1	T2	T2	T2	T2	T2	T2
P16	T1	T2	T1	T1	T1	T1	T1	T1
P17	T1	T1	T1	T1	T2	T1	T1	T2
P18	T2	T2	T1	T1	T1	T1	T1	T1
P19	T2	T2	T2	T2	T1	T2	T2	T2
P20	T2	T1	T1	T1	T1	T1	T1	T1
P21	T2	T2	T2	T2	T2	T2	T2	T2
P22	T2	T2	T2	T2	T2	T2	T2	T2
P23	T2	T2	T2	T2	T2	T2	T2	T2

Table 9: Coding of the attribute *time*. T1: Students were working at the same time; T2: Students were working at different times.

	A1	A2	A3	A4	A5	A6	A7	A8
P1	L2	L1	L2	L2	L2	L2	L2	L2
P2	L2	L2	L2	L2	L2	L2	L2	L2
P3	L2	L2	L2	L2	L1	L2	L2	L2
P4	L2	L2	L2	L2	L2	L2	L2	L2
P5	L2	L2	L2	L2	L2	L2	L2	L2
P6	L2	L1	L1	L1	L1	L2	L1	L1
P7	L1	L1	L1	L2	L2	L2	L2	L2
P8	L2	L2	L2	L2	L2	L2	L2	L2
P9	L2	L1	L2	L2	L2	L2	L2	L2
P10	L2	L2	L2	L2	L2	L2	L2	L2
P11	L2	L2	L2	L2	L2	L2	L2	L2
P12	L2	L2	L2	L2	L2	L2	L2	L2
P13	L2	L2	L2	L2	L1	L1	L2	L2
P14	L2	L2	L2	L2	L2	L2	L2	L2
P15	L2	L1	L2	L2	L2	L2	L2	L2
P16	L2	L2	L2	L2	L1	L1	L1	L2
P17	L1	L1	L1	L2	L2	L2	L2	L2
P18	L2	L2	L1	L2	L1	L1	L1	L1
P19	L2	L2	L2	L2	L2	L2	L2	L2
P20	L2	L2	L2	L2	L2	L2	L2	L2
P21	L2	L2	L2	L2	L2	L2	L2	L2
P22	L2	L2	L2	L2	L2	L2	L2	L2
P23	L2	L2	L2	L2	L2	L2	L2	L2

Table 10: Coding of the attribute *location*. L1: Students were working co-located; L2: Students were working remote.

	A1	A2	A3	A4	A5	A6	A7	A8
P1	W2	W2	W2	W2	W2	W2	W2	W2
P2	W2	W2	W2	W2	W2	W2	W2	W2
P3	W2	W2	W2	W2	W2	W2	W2	W2
P4	W2	W2	W2	W2	W2	W2	W2	W2
P5	W2	W2	W2	W2	W2	W2	W2	W2
P6	W2	W1	W1	W1	W2	W2	W1	W1
P7	W1	W2	W1	W2	W2	W2	W2	W2
P8	W2	W2	W2	W2	W2	W2	W2	W2
P9	W2	W2	W2	W2	W2	W2	W2	W2
P10	W2	W2	W2	W2	W2	W2	W2	W2
P11	W2	W2	W2	W2	W2	W2	W2	W2
P12	W2	W2	W2	W2	W2	W2	W2	W2
P13	W1	W1	W1	W1	W1	W1	W1	W1
P14	W2	W2	W2	W2	W2	W2	W2	W2
P15	W2	W2	W2	W2	W2	W2	W2	W2
P16	W2	W1	W1	W1	W1	W1	W1	W1
P17	W2	W2	W2	W1	W2	W2	W2	W2
P18	W1	W2	W2	W2	W1	W1	W1	W1
P19	W2	W2	W1	W1	W1	W2	W2	W2
P20	W2	W1	W1	W1	W1	W1	W1	W1
P21	W2	W2	W1	W2	W2	W2	W1	W2
P22	W2	W2	W2	W2	W2	W2	W2	W2
P23	W1	W2	W1	W2	W1	W2	W2	W1

Table 11: Coding of the attribute *workflow*. `W1`: The pair processed the assignment in one go; `W2`: The pair processed the assignment in multiple sessions.

	A1	A2	A3	A4	A5	A6	A7	A8
P1	D1	D1	D1	D1	D1	D1	D1	D2
P2	D2	D2	D2	D2	D2	D2	D2	D2
P3	D1	D1	D1	D1	D2	D1	D1	D1
P4	D1	D2	D2	D2	D2	D2	D2	D2
P5	D2	D2	D2	D2	D2	D2	D2	D2
P6	D1	D1	D1	D1	D1	D1	D1	D1
P7	D1	D1	D1	D2	D2	D2	D2	D2
P8	D1	D1	D1	D1	D1	D1	D1	D1
P9	D2	D1	D2	D2	D2	D2	D2	D2
P10	D2	D1	D2	D1	D1	D1	D1	D1
P11	D2	D2	D2	D1	D2	D1	D2	D2
P12	D2	D2	D2	D2	D2	D2	D2	D2
P13	D2	D1	D1	D1	D2	D2	D2	D2
P14	D1	D2	D2	D1	D1	D1	D2	D1
P15	D2	D2	D2	D2	D2	D1	D2	D2
P16	D1	D1	D1	D1	D1	D1	D1	D1
P17	D1	D1	D2	D2	D1	D2	D2	D2
P18	D1	D2	D2	D2	D1	D1	D1	D1
P19	D2	D2	D1	D2	D2	D2	D2	D2
P20	D2	D2	D1	D1	D1	D1	D1	D1
P21	D2	D2	D1	D2	D1	D2	D2	D2
P22	D1	D1	D1	D2	D1	D1	D1	D2
P23	D1	D1	D1	D1	D1	D2	D2	D1

Table 12: Coding of the attribute *documents*. D1: Only one codestrate was used per pair; D2: Multiple codestrates were used per pair.

	A1	A2	A3	A4	A5	A6	A7	A8
P1	S1	S1	S3	S3	S3	S3	S3	S3
P2	S3	S3	S3	S3	S3	S3	S3	S3
P3	S1	S1	S1	S1	S1	S3	S3	S1
P4	S4	S1	S1	S1	S1	S1	S1	S1
P5	S1	S1	S1	S1	S1	S1	S1	S1
P6	S1	S1	S1	S1	S1	S4	S1	S1
P7	S1	S1	S3	S1	S3	S3	S3	S2
P8	S3	S2	S3	S3	S4	S3	S3	S3
P9	S3	S1	S1	S1	S3	S3	S3	S3
P10	S4	S1	S1	S1	S1	S1	S1	S1
P11	S1	S1	S1	S1	S1	S1	S3	S1
P12	S1	S1	S1	S1	S1	S1	S1	S1
P13	S1	S1	S1	S1	S1	S3	S1	S3
P14	S1	S2	S2	S2	S2	S2	S2	S2
P15	S4	S1	S1	S1	S2	S1	S1	S1
P16	S1	S2	S1	S1	S1	S1	S1	S2
P17	S1	S1	S1	S3	S3	S3	S3	S1
P18	S4	S1	S1	S3	S1	S2	S1	S1
P19	S1	S1	S1	S1	S1	S1	S1	S1
P20	S4	S1	S1	S1	S1	S1	S1	S1
P21	S4	S1	S1	S1	S1	S1	S1	S1
P22	S4	S4	S4	S3	S3	S3	S3	S3
P23	S2	S4	S3	S2	S3	S3	S3	S3

Table 13: Coding of the attribute *work split*. S1: All parts of the assignment were done together; S2: The assignment was split by part (HTML, CSS, JavaScript); S3: The assignment was split by topic; S4: No questionnaire was filled out.

BIBLIOGRAPHY

Badam, Sriram Karthik, Andreas Mathisen, Roman Rädle, Clemens N. Klokmose, and Niklas Elmqvist (2018). "Vistrates: A Component Model for Ubiquitous Analytics." In: *IEEE Transactions on Visualization and Computer Graphics*. ISSN: 1077-2626. DOI: 10.1109/TVCG.2018.2865144.

Beaudouin-Lafon, Michel (2000). "Instrumental Interaction: An Interaction Model for Designing Post-WIMP User Interfaces." In: *Proceedings of the 18th international conference on Human factors in computing systems - CHI '00* 2.1, pp. 446–453. ISSN: 1581132166. DOI: 10.1145/332040.332473.

Beaudouin-Lafon, Michel and Wendy E. Mackay (2000). "Reification, Polymorphism and Reuse: Three Principles for Designing Visual Interfaces." In: *Proceedings of the working conference on advanced visual interfaces* May, pp. 102–109. DOI: 10.1145/345513.345267.

Bødker, Susanne (1987). "Through the Interface - a Human Activity Approach to User Interface Design." In: *DAIMI Report Series*. ISSN: 2245-9316. DOI: 10.7146/dpb.v16i224.7586.

Bødker, Susanne (1989). "A Human Activity Approach to User Interfaces." In: *Human–Computer Interaction* 4.3, pp. 171–195. ISSN: 0737-0024. DOI: 10.1207/s15327051hci0403_1.

Bødker, Susanne and Clemens N. Klokmose (2011). "The Human-Artifact Model–an Activity Theoretical Approach to Artifact Ecologies." In: *Human–Computer Interaction* 26.4, pp. 315–371. ISSN: 0737-0024. DOI: 10.1080/07370024.2011.626709.

Borowski, Marcel, Roman Rädle, and Clemens N. Klokmose (2018). "Codestrate Packages: An Alternative to "One-Size-Fits-All" Software." In: *CHI EA '18 Proceedings of the 2018 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. DOI: 10.1145/3170427.3188563.

Butz, Andreas and Antonio Krüger (2014). *Mensch-Maschine-Interaktion*. De Gruyter Oldenbourg. ISBN: 978-3-486-71621-4.

Conlen, Matthew and Jeffrey Heer (2018). "Idyll: A Markup Language for Authoring and Publishing Interactive Articles on the Web." In: *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*. UIST '18. Berlin, Germany: ACM, pp. 977–989. ISBN: 978-1-4503-5948-1. DOI: 10.1145/3242587.3242600.

Fiala, Jakub, Matthew Yee-King, and Mick Grierson (2016). "Collaborative coding interfaces on the Web." In: *Proceedings of the 2016 International Conference on Live Interfaces*, pp. 49–58.

Hamrick, Jessica B. (2016). "Creating and Grading IPython/Jupyter Notebook Assignments with NbGrader." In: *Proceedings of the 47th*

ACM Technical Symposium on Computing Science Education. SIGCSE '16. Memphis, Tennessee, USA: ACM, pp. 242–242. ISBN: 978-1-4503-3685-7. DOI: 10.1145/2839509.2850507.

Isenberg, Petra, Danyel Fisher, Sharoda A. Paul, Meredith Ringel Morris, Kori Inkpen, and Mary Czerwinski (May 2012). "Co-Located Collaborative Visual Analytics Around a Tabletop Display." In: *IEEE Transactions on Visualization and Computer Graphics* 18.5, pp. 689–702. ISSN: 1077-2626. DOI: 10.1109/TVCG.2011.287.

Kery, Mary Beth, Marissa Radensky, Mahima Arya, Bonnie E. John, and Brad A. Myers (2018). "The Story in the Notebook: Exploratory Data Science Using a Literate Programming Tool." In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. CHI '18. Montreal QC, Canada: ACM, 174:1–174:11. ISBN: 978-1-4503-5620-6. DOI: 10.1145/3173574.3173748.

Klokmose, Clemens N. and Michel Beaudouin-Lafon (2009). "VIGO: Instrumental Interaction in Multi-Surface Environments." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 869–878. DOI: 10.1145/1518701.1518833.

Klokmose, Clemens N. and Pär-Ola Zander (2010). "Rethinking Laboratory Notebooks." In: *Proceedings of COOP 2010*, pp. 119–140. DOI: 10.1007/978-1-84996-211-7.

Klokmose, Clemens N., James R. Eagan, Siemen Baader, Wendy E. Mackay, and Michel Beaudouin-Lafon (2015). "Webstrates: Shareable Dynamic Media." In: *UIST '15 Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, pp. 280–290. DOI: 10.1145/2807442.2807446.

Kluyver, Thomas, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, and Carol Willing (2016). "Jupyter Notebooks — a publishing format for reproducible computational workflows." In: *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pp. 87–90. DOI: 10.3233/978-1-61499-649-1-87.

Knuth, Donald E. (1984). "Literate Programming." In: *Computers and Chemical Engineering* 22.12, pp. 1745–1747. ISSN: 0098-1354. DOI: 10.1016/S0098-1354(98)00029-5.

Norman, Donald (2013). *The Design of Everyday Things: Revised and Expanded Edition*. Basic Books. ISBN: 9780465072996.

Nouwens, Midas and Clemens N. Klokmose (2018). "The Application and Its Consequences for Non-Standard Knowledge Work." In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems - CHI '18*, pp. 1–12. DOI: 10.1145/3173574.3173973.

O'Hara, Keith J., Doug Blank, and James Marshall (2015). "Computational Notebooks for AI Education." In: *Twenty-Eighth International Florida Artificial Intelligence Research Society Conference (FLAIRS)*. DOI: 10.13140/2.1.2434.5928.

Olsen Jr., Dan R. (2007). "Evaluating User Interface Systems Research." In: *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*. UIST '07. Newport, Rhode Island, USA: ACM, pp. 251–258. ISBN: 978-1-59593-679-0. DOI: 10.1145/1294211.1294256.

Olson, Judith S., Dakuo Wang, Gary M. Olson, and Jingwen Zhang (Mar. 2017). "How People Write Together Now: Beginning the Investigation with Advanced Undergraduates in a Project Course." In: *ACM Trans. Comput.-Hum. Interact.* 24.1, 4:1–4:40. ISSN: 1073-0516. DOI: 10.1145/3038919.

Pérez, Fernandez (Apr. 2013). *"Literate computing" and computational reproducibility: IPython in the age of data-driven journalism*. Blog. (accessed November 15, 2018). URL: http://blog.fperez.org/2013/04/literate-computing-and-computational.html.

Rädle, Roman, Midas Nouwens, Kristian Antonsen, James R. Eagan, and Clemens N. Klokmose (2017). "Codestrates: Literate Computing with Webstrates." In: *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. UIST '17. Québec City, QC, Canada: ACM, pp. 715–725. ISBN: 978-1-4503-4981-9. DOI: 10.1145/3126594.3126642.

Rule, Adam, Aurélien Tabard, and James D. Hollan (2018). "Exploration and Explanation in Computational Notebooks." In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. CHI '18. Montreal QC, Canada: ACM, 32:1–32:12. ISBN: 978-1-4503-5620-6. DOI: 10.1145/3173574.3173606.

ShareLaTeX Blog (Mar. 2017). *Track changes and comments in ShareLaTeX*. Blog. (accessed November 15, 2018). URL: https://www.sharelatex.com/blog/2017/03/09/track-changes-and-comments-in-latex.html.

Shneiderman, Ben (1983). "Direct Manipulation: A Step Beyond Programming Languages." In: *Computer* 16.8, pp. 57–69. ISSN: 0018-9162. DOI: 10.1109/MC.1983.1654471.

Srnec, Matthew N., Shiv Upadhyay, and Jeffry D. Madura (2016). "Teaching Reciprocal Space to Undergraduates via Theory and Code Components of an IPython Notebook." In: *Journal of Chemical Education* 93.12, pp. 2106–2109. DOI: 10.1021/acs.jchemed.6b00392.

Suwantarathip, Ornprapat and Saovapa Wichadee (2014). "The Effects of Collaborative Writing Activity Using Google Docs on Students' Writing Abilities." In: *Turkish Online Journal of Educational Technology - TOJET* 13.2005, pp. 148–156. ISSN: ISSN-1303-6521.

Tang, Anthony, Melanie Tory, Barry Po, Petra Neumann, and Sheelagh Carpendale (2006). "Collaborative Coupling over Tabletop Displays." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '06. Montréal, Québec, Canada: ACM, pp. 1181–1190. ISBN: 1-59593-372-7. DOI: 10.1145/1124772.1124950.

Tchernavskij, Philip, Clemens N. Klokmose, and Michel Beaudouin-Lafon (2017). "What Can Software Learn From Hypermedia?" In: *Proceedings of the International Conference on the Art, Science, and Engineering of Programming - Programming '17*. New York, New York, USA: ACM Press. ISBN: 9781450348362. DOI: 10.1145/3079368.3079408.

Tidwell, Jenifer (2010). *Designing Interfaces*. O'Reilly Media, Inc. ISBN: 1449379702, 9781449379704.

Weiser, Mark (Sept. 1991). "The Computer for the 21st Century." In: *Scientific American* 265.3, pp. 94–104. ISSN: 0036-8733. DOI: 10.1038/scientificamerican0991-94.

Wilson, Greg, Fernando Perez, and Peter Norvig (2014). "Teaching Computing with the IPython Notebook (Abstract Only)." In: *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*. SIGCSE '14. Atlanta, Georgia, USA: ACM, pp. 740–740. ISBN: 978-1-4503-2605-6. DOI: 10.1145/2538862.2539011.

Yim, Soobin, Dakuo Wang, Judith Olson, Viet Vu, and Mark Warschauer (2017). "Synchronous Collaborative Writing in the Classroom: Undergraduates' Collaboration Practices and Their Impact on Writing Style, Quality, and Quantity." In: *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*. CSCW '17. Portland, Oregon, USA: ACM, pp. 468–479. ISBN: 978-1-4503-4335-0. DOI: 10.1145/2998181.2998356.

Zhou, Wenyi, Elizabeth Simpson, and Denise Pinette Domizi (2012). "Google Docs in an Out-of-Class Collaborative Writing Activity." In: *International Journal of Teaching and Learning in Higher Education* 24.3, pp. 359–375. ISSN: 1812-9129.