Report from Dagstuhl Seminar 22302

Educational Programming Languages and Systems

Neil Brown[*1], **Mark J. Guzdial**[*2], **Shriram Krishnamurthi**[*3], **and Jens Mönig**[*4]

1 **King's College London, GB.** dagstuhl@twistedsquare.com
2 **University of Michigan – Ann Arbor, US.** mjguz@umich.edu
3 **Brown University – Providence, US.** shriram@gmail.com
4 **SAP SE – Walldorf, DE.** jens@moenig.org

Abstract

Programming languages and environments designed for educating beginners should be very different from those designed for professionals. Languages and environments for professionals are usually packed with complex powerful features, with a focus on productivity and flexibility. In contrast, those designed for beginners have quite different aims: to reduce complexity, surprise, and frustration.

Designing such languages and environments requires a mix of skills. Obviously, some knowledge of programming language issues (semantics and implementation) is essential. But the designer must also take into account human-factors aspects (in the syntax, development environment, error messages, and more), cognitive aspects (in picking features, reducing cognitive load, and staging learning), and educational aspects (making the language match the pedagogy). In short, the design process is a broad and interdisciplinary problem.

In this Dagstuhl Seminar we aimed to bring together attendees with a wide variety of expertise in computer education, programming language design and human-computer interaction. Because of the diverse skills and experiences needed to create effective solutions, we learned from each other about the challenges – and some of the solutions – that each discipline can provide.

Our goal was that attendees could come and tell others about their work and the interesting challenges that they face – and solutions that they have come up with. We aimed to distill lessons from the differing experiences of the attendees, and record the challenges that we jointly face. The seminar allowed attendees to share details of their work with each other, followed by discussions, and finally some plenary sessions to summarize and record this shared knowledge.

Seminar July 24–29, 2022 – http://www.dagstuhl.de/22302
2012 ACM Subject Classification Applied computing → Education; Software and its engineering → Software notations and tools
Keywords and phrases computer science education research, errors, learning progressions, programming environments
Digital Object Identifier 10.4230/DagRep.12.07.205

1 Summary

Mark Guzdial

To the world at large, programming is one of the most visible and valuable aspects of computing. Yet learning to program has been well documented as challenging and a barrier to entry. The way that a computer interprets a program literally and the complex

[*] Editor / Organizer

Educational Programming Languages and Systems, *Dagstuhl Reports*, Vol. 12, Issue 07, pp. 205–236
Editors: Neil Brown, Mark J. Guzdial, Shriram Krishnamurthi, and Jens Mönig
Dagstuhl Reports
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

interdependencies within programs are surprising to novices, who have a rather limited understanding of things that can go wrong in programs and what can be done to protect against these problems.

Thinking about computing (and specifically programming) education is particularly timely for several reasons:

- Numerous countries, states, and other geographic entities are making a big push to put computing into curricula for all students.
- Computing tools are now making a serious dent into several disciplines, not just traditional sciences (like physics and biology) but also in social sciences (like history and sociology). New hybrid areas like bioinformatics and data science are being created. People in numerous disciplines now would benefit from, and in some cases need to, learn how to program.
- People outside traditional academic structures are being given the power to program. Everything from spreadsheets to home automation systems are providing "scripting" interfaces that people can use to simplify or enrich their lives. At the same time, those who do not adapt to these trends risk being left behind in their jobs.

In response to the challenges that programmers face, computer scientists have created numerous programming languages and systems, such as interactive development environments (IDEs). To a professional, the more the tools, usually, the better. Tools that can compute advanced analyses, whose meaning may take a great deal of training to understand (as just one example, a worst-case execution time analysis to aid in building a real-time system, or a dependent type system that can statically analyze rich program invariants) can be well worth the investment.

Beginners, however, have a very different set of concerns. For instance:

Syntax Basic matters of syntax can be problematic. Beginners can struggle with the notion that the computer requires very precise utterances (something they have not had to deal with before in writing, math, and other disciplines, where a human reader is usually able to deal with ambiguity, and is forgiving). They may also struggle with basic tasks like typing.

Graduated Introduction A typical textbook will introduce concepts slowly and gradually: each chapter, for instance, might introduce one new concept. Our tools, however, do not typically offer this same graduation. Instead, a student put in front of (say) a Java IDE must confront all of Java: a typo could result in an indecipherable error, strange behavior, and so on. This can result in a very confusing, and unfriendly, learning environment.

Errors Computing is relatively unique in confronting learners with a large number of errors. But errors are intimidating to beginners (many of whom worry that they can "break the computer"), and can at any rate be a deflating experience. At the same time, errors can be viewed as a learning opportunity. How do we design errors – and more broadly, system feedback – in such a way that it is constructive, comprehensible, and encouraging?

Accessibility Computing has traditionally had a rather shameful relationship with users who have special needs. For example, the number of blind professional developers is negligible, and in not at all representative of the percentage of blind people in the population. However, spending a few minutes with our programming tools will make clear why this is not surprising at all: so much is oriented towards visual inspection and manipulation. Similarly, our tools are rarely tested against, for instance, the needs and capabilities of learners with cognitive impairments.

While it may take a long time before the entire computing "pipeline" can adjust to such needs, we can still start to make progress in this direction. Furthermore, the community of beginners who do not need much computing sophistication can immediately benefit from advances in this area.

Our seminar was successful in bringing together attendees with expertise in computing education, programming language design, and human-computer interaction. The presentation and discussions explored a wide range of issues, from language and environment design, to teaching methods and assessment issues. In particular, we studied and discussed:

Tools and Languages A wide variety of tools and languages were presented – from block-based to textual, from imperative to functional, to those embedded in games, and to those explicitly designed for programmer's whose native language is not English. Some of the tools were explicitly for learning and teaching of programming, like tutors and ebooks. We discussed tools for understanding program execution, such as debuggers.

Blocks Many of the attendees are exploring block-oriented programming in some way, so we had several sessions focused on novel uses of block-based programming and applying the blocks modality in different kinds of programming languages and paradigms.

Data Increasingly, we recognize that learning programming is not just about the language and the IDE, but also about *data* – what data students use and how data are described and structured. Data can be motivating for students. Carefully selected data sets can play an important role in supporting learning in contexts outside of computing, e.g., about social issues or about scientific phenomena). Data can be complex and messy, which can take more time to explain and "clean."

Learning Issues Our discussion included the cognitive issues when learning programming and the challenges in helping students to transfer knowledge to new contexts. We discussed the strategies to be taught to students to help them succeed at reading, writing, and debugging programs. We saw teaching techniques that are unique to programming, like Parsons problems. We particularly focused on the cognitive tasks of planning and identifying goals, which are critical to student success in programming. We discussed learning trajectories that explain how we might expect student learning to occur.

Process Computer science has been called the study of algorithmic processes, and computer science education research needs to also be concerned with students learning computational processes in the context of other processes. Some of the processes we discussed include learning processes (i.e., what has to happen to make sure that learning is successful and is retained and possibly transferred), classroom processes (i.e., how does programming fit into the classroom context, including how work is evaluated), and student programming processes (i.e., students are learning design, development, and debugging processes, and need scaffolding to help them succeed and develop their processes to be more expert-like).

Practical Issues of Sustainability Building software has always been difficult to fit into most national research infrastructures. Funding agencies are often reticent to pay for software development, let alone maintenance. Maintaining software over time is expensive but is critical to test educational hypotheses in ecologically-valid contexts (i.e., classrooms vs laboratories) and to gain the benefits of novel software implementations in education. A key area of maintenance of educational software is the fit between the curriculum, the teacher, the school context, and the software. Software may need to change as teaching goals and teachers change, which is a new and complex area for software maintenance.

2 Table of Contents

3 Overview of Talks

3.1 OCaml Blockly

Kenichi Asai (Ochanomizu University – Tokyo, JP)

Joint work of Kenichi Asai, Haruka Matsumoto

OCaml Blockly is a block-based environment for OCaml, built on top of Google Blockly. It has the following features:

- It produces no syntax error, once all the holes are filled with blocks.
- It produces no unbound variable error, since the scoping rules of OCaml are built in. Variable blocks can exist only at the legal places.
- It produces no type error, since the typing rules of OCaml are built in.

In a nutshell, OCaml Blockly behaves decently as the language designer expects. It has been used seriously since 2019 in a functional language course for CS-major students as well as an introductory game programming course for non-CS-major students and one-day seminars for high school students.

3.2 OCaml Stepper

Kenichi Asai (Ochanomizu University – Tokyo, JP) and Youyou Cong (Tokyo Institute of Technology, JP)

Joint work of Tsukino Furukawa, Hinano Akiyama, Kenichi Asai, Youyou Cong
Main reference Tsukino Furukawa, Youyou Cong, Kenichi Asai: "Stepping OCaml", in Proc. of the Proceedings Seventh International Workshop on Trends in Functional Programming in Education, TFPIE@TFP 2018, Chalmers University, Gothenburg, Sweden, 14th June 2018, EPTCS, Vol. 295, pp. 17–34, 2018.
URL http://dx.doi.org/10.4204/EPTCS.295.2

OCaml Stepper is an algebraic stepper for OCaml that supports the basic constructs of OCaml as well as exception handling, outputs, and modules. Since it produces each step incrementally, it can step execute non-terminating programs. It is used in a functional language course for CS-major students to help the students understand the behavior of programs, in particular, how recursion works, where an infinite loop occurs, and where programs exhibit unintended behaviors.

3.3 Snap! Tools for Customizing The Environment For Teachers and Learners

Michael Ball (University of California – Berkeley, US)

Joint work of Michael Ball, Jens Mönig, Brian Harvey

Snap! has broad support for making customizations to the programming environment that aid in teaching students. Most of these tools are designed for instructors to build Snap! Projects that they then give to students, rather than something students use themselves. Snap! has

always had the ability to make custom control structures which aid in creating "Domain Specific Languages" that may be easier for students to learn. Recently, we have expanded the suite of tools that allow instructors to create "microworlds" which direct the student's attention to a subset of blocks [1]. Finally, I'll show how we can use metaprogramming in Snap! to write code that dynamically changes the environment, such as adding or hiding blocks. This allows instructors to add "levels" or different modes to projects.

3.4 Columnal

Neil Brown (King's College London, GB)

URL https://www.columnal.xyz/

Data processing is an important use of computers. Spreadsheets have issues with usability and flexibility, while programming languages like R can be confusing for novices and can obscure the data which should be the central focus. Columnal is a tool that on the surface appears like a spreadsheet, but underneath has a pure functional programming model that manipulates tables in a processing pipeline. The transformations are made visible and fully reproducible in a visual system.

3.5 Structured expressions

Neil Brown (King's College London, GB)

Main reference Michael Kölling, Neil C. C. Brown, Amjad Altadmri: "Frame-Based Editing", Journal of Visual Languages and Sentient Systems, Vol. 3, pp. 40–67, KSI Research Inc., 2017.
URL http://www.ksiresearch.org/vlss/journal/VLSS2017/vlss-2017-kolling-brown-altadmri.pdf

Program source code can often be separated into statements and expressions. Statements are well-suited to being represented as draggable items in block-based programming. However, expressions are less well suited to mouse manipulation as blocks. In this talk I showed an alternative approach, from our "Stride" tool, which structures expressions more at the lexing level than the parsing level, and supports keyboard entry while maintaining the structured nature that eliminates many syntax errors.

3.6 Mio: A Block/Text Programming Environment

Youyou Cong (Tokyo Institute of Technology, JP)

Joint work of Youyou Cong, Junya Nose, Hidehiko Masuhara

The program design recipe is a sequence of steps for defining functions. It serves as an excellent guidance for beginning students, but students tend to skip intermediate steps and go straight to coding, either because they think those steps are not important, or because

they do not know what to write at each step. We propose Mio, an environment with built-in support for design-recipe-based programming. Mio provides blocks for steps before coding, and generates feedback on those steps. A preliminary experiment shows that the blocks and feedback can be effective in encouraging students to follow the recipe.

3.7 Ask-Elle and other model-based tutors

Bastiaan Heeren (Open University – Heerlen, NL) and Johan Jeuring (Utrecht University, NL)

Joint work of Bastiaan Heeren, Johan Jeuring, Alex Gerdes, Thomas Binsbergen, Hieke Keuning, Josje Lodder, Niek Mulleners
Main reference Alex Gerdes, Bastiaan Heeren, Johan Jeuring, L. Thomas van Binsbergen: "Ask-Elle: an Adaptable Programming Tutor for Haskell Giving Automated Feedback", Int. J. Artif. Intell. Educ., Vol. 27(1), pp. 65–100, 2017.
URL http://dx.doi.org/10.1007/s40593-015-0080-x

Ask-Elle is a tutor for learning the higher-order, strongly-typed functional programming language Haskell. It supports the stepwise development of Haskell programs by verifying the correctness of incomplete programs, and by providing hints. Programming exercises are added to Ask-Elle by providing a task description for the exercise, one or more model solutions, and properties that a solution should satisfy. The properties and model solutions can be annotated with feedback messages, and the amount of flexibility that is allowed in student solutions can be adjusted. The main contribution of our work is the design of a tutor that combines (1) the incremental development of different solutions in various forms to a programming exercise with (2) automated feedback and (3) teacher-specified programming exercises, solutions, and properties. The main functionality is obtained by means of strategy-based model tracing and property-based testing. We have tested the feasibility of our approach in several experiments, in which we analyse both intermediate and final student solutions to programming exercises. Similar techniques have been used in tutoring systems for code refactoring and for proofs with structural induction.

3.8 Towards Giving Timely Feedback to Novice Programmers

Johan Jeuring (Utrecht University, NL)

Joint work of Johan Jeuring, Hieke Keuning, Samiha Marwan, Dennis Bouvier, Cruz Izu, Natalie Kiesler, Teemu Lehtinen, Dominic Lohr, Andrew Petersen, Sami Sarsa

Every year, millions of students learn how to write programs. Learning activities for beginners almost always include programming tasks that require a student to write a program to solve a particular problem. When learning how to solve such a task, many students need feedback on their previous actions, and hints on how to proceed. For tasks such as programming, which are most often solved step by step, the feedback should take the steps a student has taken towards implementing a solution into account, and the hints should help a student to complete or improve a possibly partial solution. Research on feedback addresses questions about when feedback should be given, why, and how. This talk discusses how this research

is translated to when, why and how to give feedback on, or a hint at, a particular step a student takes when solving a programming task. We select datasets consisting of sequences of steps students take when working towards a solution to a programming problem, and annotate these datasets at those places at which we think an expert should intervene, why the expert intervenes, and how the expert wants to intervene. We use these datasets to compare feedback and hints given by learning environments for programming to the hints and feedback in the annotated datasets.

3.9 TigerJython and Its Debugger

Tobias Kohn (Utrecht University, NL)

Joint work of Tobias Kohn, Bill Manaris, Aegidius Plüss, Jarka Arnold
Main reference Tobias Kohn, Bill Z. Manaris: "Tell Me What's Wrong: A Python IDE with Error Messages", in Proc. of the 51st ACM Technical Symposium on Computer Science Education, SIGCSE 2020, Portland, OR, USA, March 11-14, 2020, pp. 1054–1060, ACM, 2020.
URL http://dx.doi.org/10.1145/3328778.3366920

Development environments mediate between the programmer and the machine. Aimed at novice programmers, TigerJython is a Python environment that seeks to improve, in particular, the programmer's interaction with the executing interpreter. It separates I/O more clearly from program code, displays enhanced error messages and provides a debugger for program visualisation, but also deliberately refrains from more complex features such as project management. A key insight for the design was to base the various features on a consistent and novice-friendly mental model of the machine.

3.10 The Materials and Media of Programming

Shriram Krishnamurthi (Brown University – Providence, US)

Joint work of Shriram Krishnamurthi, Elijah Rivera, Rob Goldstone, John B. Clements, Greg Cooper, Kathi Fisler, Justin Pombrio

What materials and media can we reach for to practice and learn programming? Many present themselves to our attention. Over multiple talks, we sampled several of these:

- Program planning using a mix of code and text (thanks, Snap*!*)
- Multiple notional machines based on stacks and trees
- New models of reactive programming, especially ones that attend to a diversity of educational and intellectual needs beyond just the production of output
- The exploration of variation in semantics through bottom-up discovery followed by top-down theory formation

3.11 Cognitive skills and learning to program – are meta-analyses of transfer effect useful?

Eva Marinus (Pädagogische Hochschule Schwyz, CH)

There have been past (Liao & Brigth, 1991) and more recent (Scherer et al., 2019) attempts to conduct meta-analyses on the transfer effects of learning to program on the acquisition of near (e.g., programming, programming conceptions) and far cognitive skills (e.g., mathematics, reasoning, creativity). These meta-analyses have reported overall transfer effect sizes of 0.41 and 0.49 respectively. However, when taking a closer look at the results, it becomes clear that it is premature to draw such conclusions as many studies on which the meta-analyses are based suffer from methodological problems, such as small sample size, the absence of an alternative treatment control group, and the absence of a pretest before the intervention. In addition, a couple of unexpected findings are reported, such as very high effect sizes (>2.0 or even >3.0), no effect of treatment duration and the fact that the largest effect sizes were found for transfer to creative thinking. It is concluded that future meta-analyses should first evaluate the quality of the studies on which the analyses are based and consider if there are enough appropriate studies to conduct the meta-analysis on.

3.12 Research Potpourri

Janet Siegmund (TU Chemnitz, DE)

Joint work of Janet Siegmund, Norman Peitek, Sven Apel, André Brechmann

Combining eye tracking and fMRI measures to disentangle the effects of single pieces of code on comprehension showed that descriptive identifiers activate natural-language areas more than nonsense identifiers. Kids in elementary school can already learn basic structures of programming with an offline Scratch Junior version. Mental health in academia is a huge problem and needs more attention. Can we improve on that to a Star-Trek like future?

3.13 Conceptual transfer in students learning new programming languages

Ethel Tshukudu (University of Botswana – Gaborone, BW)

Joint work of Ethel Tshukudu, Quintin Cutts
Main reference Ethel Tshukudu, Quintin I. Cutts: "Understanding Conceptual Transfer for Students Learning New Programming Languages", in Proc. of the ICER 2020: International Computing Education Research Conference, Virtual Event, New Zealand, August 10-12, 2020, pp. 227–237, ACM, 2020.
URL http://dx.doi.org/10.1145/3372782.3406270

The research investigates how students transfer conceptual knowledge between programming languages (Python and Java) during code comprehension. A Model of Programming Language Transfer for relative novice programmers that is based on code comprehension was developed.

Through validating the model, the research concludes that similarities between programming languages play a significant role in semantic and conceptual transfer between programming languages. This research also shows how the model was used to shape the design of a transfer pedagogy in the classroom. It revealed how the pedagogy can lead to improved conceptual transfer and understanding.

4 Additional Talk Abstracts

4.1 Ebooks

Barbara Ericson (University of Michigan – Ann Arbor, US)

Interactive ebooks are increasingly replacing traditional textbooks. There is evidence that they improve student satisfaction and learning and can be used to identify struggling students. There are commercial ebook platforms like Zybooks and several free and open-source ebook platforms including Runestone, Open DSA, and CS Circles. Runestone has over 30 ebooks for computing and math. These ebooks can contain text, images, videos, editable and runnable code, and many other types of practice problems including: Parsons problems, clickable code, and fill-in-the-blank problems. Runestone also includes a spaced practice tool. Instructors can create a custom course, create assignments, create timed exams, automatically grade assignments, and visualize student progress.

4.2 Parsons Problems

Barbara Ericson (University of Michigan – Ann Arbor, US)

Parsons problems are fragments of mixed-up code that have to be placed in the correct order. They were originally designed to provide engaging practice that constrains the logic, allow common errors, model good code, and provide immediate feedback. I have conducted research that provides evidence that Parsons problems have lower cognitive load and are significantly faster to solve than writing the equivalent code or fixing the equivalent code as long as the solution to the Parsons problem matches the most common student written solution. Most teachers and undergraduate students find them useful for learning programming, but some would rather write the equivalent code. We added the ability to toggle from a Parsons problem to the equivalent write code problem and are also testing using Parsons problems to help students who struggle while writing code. I invented two types of adaptation for Parsons problems which change the difficulty of a problem based on the learner's performance. Learners are nearly twice as likely to correctly solve adaptive Parsons problems than non-adaptive ones. Parsons problems can also be used on exams. Scores on Parsons problems highly correlate with scores on write code problems and there is evidence that they are more sensitive to student learning.

4.3 Peer Instruction

Barbara Ericson (University of Michigan – Ann Arbor, US)

Peer Instruction is a pedagogical technique to improve learning in lecture through active and collaborative learning. In Peer Instruction the instructor may assign pre-reading with an assessment before or at the beginning of lecture and then during lecture stop every 10-15 minutes to display a hard question. Students answer individually, discuss their answer with peers, and then answer again. Instructors lead a discussion about the question. Peer Instruction has led to twice the learning gains over traditional lecture in Physics and has increased student engagement and understanding in many fields including computer science. We have been adding support for Peer Instruction to the Runestone ebook platform to make it easier to run Peer Instruction sessions, find peer instruction questions, test research questions, and improve peer instruction questions over time. The tool supports synchronous in-person and synchronous remote users (via a chat interface) and asynchronous users via a saved chat.

4.4 Task plans and implementation choices

Kathi Fisler (Brown University – Providence, US)

Programming problems generally admit multiple solutions, which can differ in choices of data structures, control structures, or the orders in which lower-level steps are performed. Research in other settings suggests that people develop more flexible design skills from seeing multiple solutions to the same problem. We suggest that subtasks can be a useful mechanism for helping students contrast different solutions to the same problem. We demonstrate this with problems that involve compositions of multiple list-manipulation tasks. We show how different mappings from each task to either being computed in a separate function or being tracked in a variable summarizes high-level differences in solution structures. These mappings highlight how different implementation choices lead to different solutions within the same task structure.

4.5 HOFs as abstractions over code and examples

Kathi Fisler (Brown University – Providence, US)

A higher-order function can take a function as an input parameter to another function. Common higher-order functions include filter, map, and sort (taking the sort criterion as an input). How do we teach students what HOFs are and how they come about? One approach is to show students two functions that perform the same core action (such as filtering), then abstract over the common code to define the filter function. We have been exploring a

different approach based on teaching students the features of different higher-order function. We present students with a set of input/output pairs and ask them to classify or cluster the pairs into ones that could be produced by the same function. This is designed to help students recognize the features of higher-order list functions (such as relative lengths of input/output elements, relative types of inputs/outputs, whether all input elements are preserved). We hypothesize that learning these features might help students better understand what different higher-order functions do and when they might apply.

4.6 2D tables for introductory programming

Kathi Fisler (Brown University – Providence, US)

What's the first data structure that we show students? Many courses use arrays, lists, or classes (as tuples of fields). We propose that 2D tables, akin to spreadsheets or CSV files, are a compelling first data structure for teaching computer science (whether to majors or non-majors). Tables are authentic and pervasive. Tables lend themselves to data from a wide range of domains, some of which raise challenges around socially-responsible computing. Programming languages that support tables provide operations such as sorting, filtering, and computing; these can be taught before showing manual iteration or recursion. This choice thus lets us focus on practical tasks and issues from early in a course. We hypothesize that this will help reach populations of students who care less about programming for programming's sake, but without sacrificing depth of computing content.

4.7 Learning Strategies

Diana Franklin (University of Chicago, US)

Joint work of Diana Franklin, Cathy Thomas, Jean Salac, and others

When languages and tools are developed, there is an assumption of the process that learners go through to complete tasks. However, there is great variation in this process, and there are more and less effective processes. Therefore, it is imperative that we identify what those processes / steps are and develop strategies and scaffolds for those strategies so that more students can be successful in CT. For example, our strategy, TIPP&SEE, is useful for learning from example code. We saw statistically significantly stronger performance in students at academic risk in the treatment group, and their performance was comparable to students not at academic risk in the control group. The variance in TIPP&SEE classrooms was significantly smaller than the variance in control classrooms. There are many areas still ripe for benefiting from strategies, such as planning, decomposition, and debugging.

4.8 Learning Trajectories

Diana Franklin (University of Chicago, US)

Joint work of led by Katie Rich, with Carla Strickland and others

Learning trajectories are possible intermediate learning goals, dependencies between them, and activities that can build knowledge from one goal to another. The creation of the learning goals themselves and their dependencies is influenced by pedagogical approaches and education theory, such as constructivism, pieces of knowledge, and spiral conceptual ordering. Our learning trajectories start with everyday understandings of CS concepts, proceed to computational thinking ideas that apply to computation, and end with coding-specific goals.

4.9 Concept-Annotated Examples for Library Comparison

Elena Glassman (Harvard University – Allston, US)

Joint work of Litao Yan, Miryung Kim, Bjoern Hartmann, Tianyi Zhang

Programmers often rely on online resources – such as code examples, documentations, blogs, and Q&A forums – to compare similar libraries and select the one most suitable for their own tasks and contexts. However, this comparison task is often done in an ad-hoc manner, which may result in suboptimal choices. Inspired by Analogical Learning and Variation Theory, we hypothesize that rendering many concept-annotated code examples from different libraries side-by-side can help programmers (1) develop a more comprehensive understanding of the libraries' similarities and distinctions and (2) make more robust, appropriate library selections. We designed a novel interactive interface, ParaLib, and used it as a technical probe to explore to what extent many side-by-side concepted-annotated examples can facilitate the library comparison and selection process. A within-subjects user study with 20 programmers shows that when using ParaLib, participants made more consistent, suitable library selections and provided more comprehensive summaries of libraries' similarities and differences.

4.10 Three Examples of Participatory Design

Mark Guzdial University of Michigan – Ann Arbor, US)

Participatory design is a process of asking those using (or participating in) a program or piece of software to help with the design task. We are challenged to figure out how to set the stage so that participants and leaders have a shared understanding of what is to be designed, and then to have participation from the range of possible participants. I describe three examples of where I have used participatory design: In re-designing an on-line MS in CS program to involve more women, to design a data visualization tool for social studies teachers, and to design new computing education courses.

4.11 HyperCard

Mark Guzdial University of Michigan – Ann Arbor, US)

Apple introduced HyperCard with every Macintosh sold in 1987. At one point, it was likely the most used end-user programming environment in the world. It is unusual for its wordy, phrase-based grammar, and a tight integration with a set of GUI-building tools.

4.12 Teaspoon Languages

Mark Guzdial University of Michigan – Ann Arbor, US)

A Teaspoon language is a task-specific programming (TSP) language with three characteristics: It is a specification of a process for a computational agent (i.e., it really is programming), it can be used for a task that a teacher finds useful, and is learnable by a teacher in less than 10 minutes. These requirements make them very simple, and inexpensive to use in a class (e.g., the whole process of learning and using can fit into a single one hour course period). I describe three example Teaspoon languages for secondary school classes in mathematics, social studies, and engineering. Teaspoon languages may be a way to develop early skills and self-efficacy in computing.

4.13 Hedy

Felienne Hermans (Leiden University, NL)

Teaching block-based languages is popular at the elementary school age, but from the middle school age (10 to 14) kids tend to want to learn textual languages, but Python is still quite tricky for them because its syntax can cause high cognitive load. Hedy aims to make the path to using Python easier, with a gradual approach, using different language levels, so learners do not have to learn all syntax rules at once. In level 1, there is hardly any syntax at all, for example, printing is done with: `print Hello Dagstuhl!`

In every level, new syntax and concepts are added, until kids are doing a subset of Python in level 18 with conditions, loops, variables and lists. In addition to the cognitive load of syntax, in this talk we also discuss other ways in which Hedy aims to reduce cognitive load, such as built-in lesson plans aiming at broad applications of programming.

Hedy was launched in 2020 and since its creation 2.5 million Hedy programs have been created by children worldwide. Try Hedy at `www.hedy.org`.

4.14 Localizing programming languages

Felienne Hermans (Leiden University, NL)

Most programming languages are built with English keywords, and often leaning on the assumption that users will only use latin characters. However, for non-English speakers, especially for non-Latin or right to left languages, using English programming languages can be a large barrier. Hedy (`www.hedy.org`) allows users to program in any natural language, for which a number of technical challenges needed to be addressed, including (but not limited to) allowing non-Latin variable names, calculating using non-Latin numerals, right-to-left parsing and rendering and using multilingual grammars. Open challenges remain, such as production rules that deviate from the traditional English structure, f.e. In Turkish, one would write the condition before a keyword (i==0 iken) rather than before it (while i==0).

4.15 Higher Order Functions in Snap!

Jadga Hügle (SAP SE – Walldorf, DE)

Snap! is a programming language which supports the use of the Higher Order Functions MAP, KEEP (filter) and COMBINE (fold) and allows users to build their own HOFs.

Since Snap! is a blocks-based programming language, Higher Order Functions as well as lambdas are represented visually. They are ovally shaped like any other function but all contain a grey ring – the lambda – as one of their input slots.

The plain rings are part of Snap!'s Operators category and turn functions into data. To get to the return value again, they can be used as an input to the CALL block.

Functions in the rings have implicit parameters meaning all the input slots are left empty. When running the function, all empty input slots in the lambda function will be filled with the current data. In this example, the numbers from 1 to 10 are filled into both empty input slots in the + function.

```
map ( _ + _ ) over (numbers from 1 to 10)
```

In this talk, we showed how we represent lambda with the grey rings, how to recursively build MAP and then apply it to different forms of data like a table of Titanic passengers to find out the percentage of survivors per class or samples of a recording to create an Echo effect.

4.16 PLTutor

Amy J. Ko (University of Washington – Seattle, US)

Learning programming languages is hard, partly because the semantic rules that govern how programs execute in a language are often invisible. We present PLTutor, an approach to providing granular visualizations of these rules, helping learners infer them through causal

inference through a series of examples through forward and backward stepping through a program visualization, coupled with direct instruction about the rules. We evaluated learning gains among self-selected CS1 students using a block randomized lab study comparing PLTutor with Codecademy, a writing tutorial. In our small study, we find some evidence of improved learning gains on the SCS1, with average learning gains of PLTutor 60% higher than Codecademy (gain of 3.89 vs. 2.42 out of 27 questions). These gains strongly predicted midterms (R2=.64) only for PLTutor participants, whose grades showed less variation and no failures.

4.17 Ten Years of Teaching the World to Code with Gidget

Michael J. Lee (NJIT – Newark, US)

Teaching the world to code at scale is hard, in part due to the lack of access to learning materials and teachers. This talk describes Gidget – a free, online puzzle game designed to teach programming concepts by debugging faulty code – and major results from a selection of studies published over the past decade. In Gidget, learners help the eponymous robotic character complete its missions by modifying existing code that is nearly correct. The game also allows learners to run their code step-by-step to highlight how things change within the system at different levels of granularity. This encourages learners to inspect code structure and syntax at their own pace in a friendly environment. Additional features – such as frustration detection hints, embedded formative assessments, avoiding terms of violence (e.g., "remove" instead of "destroy"), choosing pro-social game motives (e.g., cleaning up a chemical spill), and automatically generated levels – have shown to be effective in greatly increasing novice programmers' self-efficacy and attitudes towards coding, and that it leads to measurable learning gains of introductory programming (CS1) concepts.

4.18 Block-Oriented Programming

Jens Mönig (SAP SE – Walldorf, DE)

Following an unwritten rule of sorts, that if something is important for a programming language it becomes (or should turn into) a first-class citizen *of* the language, programming language paradigms are often named after the "first-class citizens" they support, i.e. concepts that can be accessed as data. As "Object-Oriented" has first-class bundles of data-structured and behavior, "Functional" has reified and anonymous functions, an as of yet to be discovered "Block-Oriented" programming paradigm would feature blocks (syntax elements) as first-class citizens of the language. Snap! Now experimentally supports blocks that programmatically manipulate other blocks and can modify stacks of blocks at runtime.

5 Breakout Groups

5.1 Supporting Planning

Summary of questions we're trying to discuss / answer:

- How do we support the planning process? You get a complex problem statement. **How can you structure that thinking process before starting to code?**
- **How to teach the importance of planning?** Let them peer-review each other's designs/ programs/ documentation?
- **What parts of the verbal question gives you cues as to what data type, what control structure, etc?**
- **How do we generate subgoal labels that people would agree on?** Would it be easier to agree on functional programming labels?
- **What are the scaffolds for the different parts of the process?** Problem understanding phase. Algorithmic thinking to solve the problems. Implementation.
- **What are the phases of meta-cognitive processes: pre-, during, post (evaluating)? Can these processes be measured?**
- **How can we relate the theories we're talking about in this space to theories in HCI?**
- **How do we relate planning for program design / function design to planning for experience design or UI design or design thinking?**
- **How do we make planning a more valuable part of the planning process?** Bidirectional with coding. Planning becomes much more valuable in group projects because there, you need to identify the tasks and split up the group equitably. If you teach planning processes as part of the debugging process, then they hit the wall and then do it.
- **How can we create educational experiences so that students see the value of planning / documentation?** A long-term course in which students need to keep reusing their code. At some point in the course, you inherit someone else's code and documentation.
- **Can we extract a plan from examples that are close and modify them?** What can you extract out of your plan to find code that does pieces or something close. Kelleher has worked on creating examples, help students figure out what example is relevant, and help them modify it.
- **What tool features should we have to support these educational interventions and make it so much easier to complete if you do the planning features we've provided?**

  Potential features:
  - Define input and output pairs, provide feedback on their accuracy
  - Collaboration support – use planning as a communication mechanism for groups.

    Input / output types (and comments)

    Input / output pairs (for tasks and subtasks)

    Three categories: Complete, Incomplete, Incorrect
  - Scaffolding:
    - If they get it wrong how do we scaffold them?
    - highlight text description, model how to translate from text description to input/output and types, what are the common mistakes in that process (given the input give them the output)
    - Give them simpler inputs and have them give outputs

  - Understanding what is/ went wrong (and led them self-correct) vs. leading them to the solution.
  - Monitoring-scaffolds asking if what they are doing still aligns with their goals
  - Interface that allows them to go back and forth between planning (design), execution (code) and evaluation (testing & debugging?) phases so that they can adapt the planning if need be.
  - Detect plans from code and tests and give them a hint (e.g., that there is a misfit?) or can we detect that parts haven't been debugged/ tested?
  - Specify solutions (subtasks) and tests upfront and check if they appear in the student environment?
  - Challenge of matching tasks and solutions – how to cope with different solutions – would ML help here?
  - Monitoring if they got stuck: Finding/Distinguishing sink states (i.e., where did they get stuck) -
  - How to scaffold the design of test cases – especially the boundary conditions, error conditions,
  - Properties of valid / invalid inputs
  - Properties of valid / invalid outputs

  Perhaps other criteria – like complexity or run time – compare with optimal solution
  Task breakdown and division of labor (enter student names, then they divide tasks here)
  Input / output pairs for the subtasks
  Define function names and comments that all can see
  If you have a bug, could we have a debugger that steps through only a subset of the code – only the code that is relevant to a specific subset of the code. Can we automatically identify which is the relevant or the irrelevant code? Then we could grey out working code and highlight code that has not been verified correct yet.
  Can we ask them what the intermediate values are?
  When are they wrong about the values?
  When is the code wrong about the values?
  "Stepper 2.0", that skips the stuff that does not need to be checked :)
- Strategies
  What parts of the verbal question gives you cues as to what data type, what control structure, etc? What are the common mistakes?

5.2 Interaction between Learning Content/Curriculum with Tool-PL

Participants: **Mike Lee, Kenichi Asai, Felienne Hermanns, Jadga Huegle, Neil Brown, Jens Mönig, Mark Guzdial**

5.2.1 Re-designing the tools for the curriculum

- Should the tools help to make clearer how the program executes?
- What are the projects that we're asking students to do, and how do we structure the tool to support them?
- Do we start with the PL and then build the learning for the PL, or do we build the tasks first and build the PL to support the tasks?
- Snap (more free-form), Hedy (building for imagined tasks), Teaspoon languages (building the PL for the specific tasks)

- Building for specific purposes allows us to optimize the UI. BlueJ vs IntelliJ. How much do we have to be authentic/professional so that students buy-in?
- You don't want a game to look too professional. It has to be fun and playful. Learning is an afterthought. Does it make transfer harder later? Students seem to make the leap to Python easily. They're not surprised that the UI changes.
- Self-efficacy: They're confident in Gidget, so they think they can learn programming.
- In Germany, it's odd that you'd use a pedagogical version of anything.
- Explicitly, they're trying to make Snap into a presentation tool.

- Projects of projects, to do scenes separately and compose them.
- Make this about assembling things you like to do, not being about (serious voice) "PROGRAMMING"!

- When we teach Teaspoon languages, we say explicitly "This is programming" and "This is not." Works and not-works as you might predict. Teachers value having an explicit process that can be inspected and changed. Teachers are also scared about error messages and not being able to figure out what's wrong.
- Research Question (RQ): Can we bridge direct-manipulation and programming interfaces?

- RQ: Could the GUI tool explain itself in code with good abstraction, good semantics? Not just "click at 10,56" but grabbing an edge of an object, or clicking a button, or part of achieving a subgoal?

- Chaos game as a way to get emergent behavior. Start at a random place, pick another place, put a dot at a point halfway.

- Algorithm so simple.
- Goal: Make the programming simple to tell these kinds of simple but powerful stories.
- Can teachers do this? Education professor uses it. Works well with teacher workshop.
- Having fewer commands/blocks means fewer things will go wrong.
- Every block in Snap! Is a Teaspoon language.

- RQ: Need to collect these stories – how are the PLs changing to support specific learning scenarios?

5.2.2 Supporting Teachers and Classrooms

- Alphonse the Camel – doesn't really work in a computational timeline. Maybe some things just work better in their existing form.
- Programming tasks need to work at a level that is comfortable for students in a classroom discourse. Computational language needs to be at the right level of abstraction.

- How do we represent and talk about powerful ideas like emergence?

- How do we support teachers in doing interesting and innovative work? How do we give them the confidence to make changes, to explore, and to feel that they can succeed?
- As a teacher, you're more confident if you know ALL the surface features, all the available blocks.
- What friction do you have to pay to get started, to learn it?
- How much up-front costs are there?
- Trade-offs between limiting the space to be known and what's possible.

- Hedy reduces what is to be known, but then you can't build so much.

5.2.3 Building Curriculum

- About the stories more than the concepts.
- Western culture and educational systems value the abstract concepts over the concrete instantiations.
- We want to "Storm heaven" – change everything.
- "But does this help me pay my bills?"
- "Tyranny of the Status Quo" – trying to change everything means that someone is going to lose.
- Decompose the stories into the parts that the computer can/should do and what we should do in natural language.
- What are the possibilities for what we can do in the computer?

5.2.4 Maintaining Content and PL connections

- Keeping BlueJ updated with Java – invalidates teacher's materials.
- Building unit tests from the examples that are useful in Columnal
- It's a good example. Make sure it always work.
- Harder to update curriculum content, e.g., screenshots.
- Teachers can upload their own content into the curriculum at Hedy. Can we class-specific or shared with others.
- Easier for them.
- Less cognitive load.
- AND all examples are automatically indexable and updatable. If Felienne updates semantics, she can automatically refactor their examples and keep them updated.
- We alert them. "Warning: This is the change."
- Creates sense of community. Ownership.
- Mike Lee supports teachers in Gidget customizing and adding levels AFTER they finish the 37 built-in levels.
- Use-driven programming language design.
- Can look at teachers' materials to figure out where they want to go.
- Do teachers want to change order, e.g., variables, conditionals, and looping? Can we support that in the language?
- What if teachers spent more units on given concepts? Does that tell us what concepts are harder than we expected?
- How do you deal with privacy (student identity)? What if they say something vile or evil? How to avoid spam and manipulation of page rank?
- Snap games about shooting about schools
- Hedy tries to moderate in all their different languages.

- Snap forum is a small slice of the community. Totally slanted.

- "How to hack in JavaScript in Snap"
- "Why isn't Jens paying attention to me?"
- It's hard to give all of this the attention the community needs.

- Supporting the Computing At School in UK community on-line.

- Didn't want to turn away industry people, but focused on teachers.
- A couple of people were very negative. Wanted to ban them. It's our forum just kick them off. But you want to have fair application of rules. One person can change a community – you don't see what people don't post because they're scared.
- Felienne has lots of experiences with trolls.

- We foster a culture where being an asshole is how you get attention, how you "lead."
- Teacher culture is not the same. They tend NOT to be loud, to be mean to be a leader.
- "I don't need a professor from Georgia to tell me how to do my job."

- Teachers don't want professors telling them what to do.
- Educators (teachers and leadership) value most heavily local voices.

- CAS was primarily academics, but perceived as being mostly industry-driven.

- In part because Simon Peyton Jones was leading, and he was at Microsoft.
- But was instigated by Eric Schmidt of Google saying "Fix this!"

- In the end, teachers are our main curriculum developers.

5.2.5 How can we foster a culture of plurality of tools and concepts

- Be more tolerant of imperative vs functional, text vs blocks, abstract vs concrete.
- Get past fear that we have to do things one way and that we use up all the other space.

- "Will having more than one thing confuse teachers?"
- In the end, humans are good at filtering out what they don't need/want, and coping with complexity.
- We don't have to find the ONE way.
- It's okay to have competing visions.

- Teachers worry about teaching the wrong thing. They want to know the right thing.
- If you love what you're doing, teach that.

- Dijkstra quote: Basic ruins your brain.

- How do teachers deal with technology rapidly changing? Teachers want to develop depth in a specific tool/PL. But it keeps changing.

- RQ: We need Ethel-like work studying teachers as they transfer knowledge and curriculum and PCK into new versions and new languages. How do they do it?

- Different goals: Breadth vs. depth. Computational practices need depth.
- CS Teachers are hesitant because they don't know the answers.

- Teachers don't want to move to CS because they have to keep updating to keep up.
- In sciences, things change, too. But calculus and other subjects, changes can be about PCK and not underlying content.
- How true is that in CS? Depends on language. Java and JavaScript changes dramatically.
- "Laminating your lesson plans."

- Give teachers a way to stick with what they're teaching, e.g., VMs.

- Python print() vs print.

- Challenge: Making Snap keep looking like Scratch to be comfortable but...

- Changing underneath dramatically.
- Teacher say "There's nothing new here!"

5.3 Program Representation

Participants: **Kathi, Ethel, Michael, Tobias, Amy, Janet**

5.3.1 Synopsis

- There are many questions about why we need representations; are they a means to greater understanding or an essential part of programming and learning?
- There are also many questions about whether automated computational tools for generating representations of program behavior is essential, or just a bias we have as a discipline; is it possible that whiteboarding and sketching skills should get our attention instead?
- There are some ways that representations may benefit from social settings, using communication as a vehicle to generate needed representations as needed.
- Representations may also need to leverage prior knowledge. One example of this is having domain knowledge about the data passing through algorithms, especially personal data. This can promote greater comprehension by leveraging learners' assets.
- There may be some value in trying to name some of the many representations that we've invented and teach people how to use them flexibly to reason about algorithms and program behavior. Maybe they will be compelling to the extent they're situated in relevant domains and personal data.

5.3.2 Notes

Why are we here?

- Amy is overwhelmed by programming language design choices for grade 6-12 creative expression contexts
- Michael is trying to understand challenges with Python Tutor and challenges with a purely functional model that leads multiple executions on one line that isn't well supported by professional stepping tools. Also, Snap doesn't have a lot of tools for observing execution. What should be built into the language?
- Janet is teaching two different classes, one is CS2, one is CS1 for a masters course with students who do not have programming experience. Motivated to restructure CS2 course. Exploring different program representations.

- Tobias is wondering about the relationship between the static program and how it dynamically executes. Students want to modify a program while the debugger is running; how to support them during a debugging process. Just having values alone often does not help with understanding. Comment from Janet: Just the numbers only allows students to implicitly develop a mental model of how program execution works
- Ethel has seen PythonTutor and thought it was helpful but wants to understand reasons for choosing a representation and how representation influences transfer.
- Kathi has been wrestling with the question of why, from a learning perspective, are we creating program representations. Tracing programs isn't necessarily the skill we need students to have; we see tracing as a means to an end of broader programming skills, such as debugging, and thinking about behavior. (Some discussion amongst the group about what student goals we're actually targeting and how much the mechanics of understanding program execution actually helps with higher level understanding of program behavior.)

What is learning programming actually about?

- Ethel: code comprehension and writing skills doesn't have a strong correspondence. How do we avoid them feeling like things are important when it's the thing that's in the code. [`https://dl.acm.org/doi/10.1145/3341525.3387379` - "If They Build It, Will They Understand It? Exploring the Relationship between Student Code and Performance"]
- Kathi's goal is preventing defects. She doesn't care about correct tracing per se, just that they avoid making defects.
- Amy wondered about whether understanding is necessary for building something that works. (How important is understanding? If students have sufficient ability (in different aspects) to produce a correct program, is that enough?)
- Code writing goal in industrial context: Ship product and make money (Amy); or build something without understanding (Michael)
- For some students, there's a goal of just getting something to work, and understanding becomes important when things aren't working.
- Kathi: Aliasing and mutation is the only reason she teaches notional machines
- Tobias: understanding in industry is becoming important because of security; not just debugging.
- Michael: program representations are also helpful at showing logic errors. Sometimes it's helpful to just have a tool that helps make it clear where something went wrong.
- Amy: everyone needs different representations constantly (not all of them, not the deepest ones); representation as set of possible thinking tools with different utility in different context (lots of hand waving because of missing vocabulary); we create missing representations for thinking
- Kathi: What kind of thinking should students be doing?
- Tobias: Where does change in data happen? as important to understanding
- Representations of rainfall, such that input transforms until final result(avg); each is a different representation; Amy: Why do we need something fancy, when the whiteboard on the fly representation works so well? Maybe we should teach students to get good at generating representations on whiteboards?
- Kathi: We try to make students make debugging plans where they state what they expect memory to look like after to guide their debugging expectations.
- Amy: We see students trying to build and use their own abstract representations.
- Ethel: if students bring some understanding of the world with them, perhaps representations are a way of connecting to that prior knowledge.

- Michael: What we can grade and what we can assign points for is a tension; students have learned for 20 years of their life that are useful skills for following incentives. These skills are also often done with more than one person. Having communication involved may be an important part of representation generation skill.
- Amy: Constraints of circumstances for students learning may make it more difficult to support them
- Kathi: a lot of this depends on what kinds of programs students are creating; content can be a way of helping them build on their prior knowledge.
- Janet: metaphors of variables often do not help with programming skill.
- Amy: start with a feed of social media posts in a list; filtering, counting. . . , just like with Kathi's starting with tables as real data; personal data is even better than real data for learning effects, because the meaning of the data is so important (in ML context, in which students build classifiers for their own data)
- Michael: similar ideas in taking lists of text; using people's names and converting them into initials works similarly well. Personalizing data is powerful. Familiarity of data gives context for reasoning.
- Ethel: Aha moments happened in industry when there was context for the abstract skills they were learning.
- Amy: A tool that helps students build representations that they need; what would be the value in construction for students' learning? -> Janet: We may need a study
- Michael: What is the right time for the aha moment? The programs we teach are not set up to provide the context that leads to the aha. But projects outside of class often are.
- Tobias: The problem with the aha moment is that they are only valuable if they are emergent, not forced. You want to feel like you are a "smart cookie".
- Ethel: This may not happen for everyone; some students may not catch it the first time and may need teachers to help build the aha moment for them.
- Michael: Different projects can build these moments differently. We use a Plants vs zombie-like game, and a data-oriented project which click with different students.
- Tobias: Aha-moments are important and may come later, way after the course
- Kathi: What's the baseline we have to get them to do get them to be able to build their own representations? Have students develop an explanation for other students to understand programs/concepts
- Amy: Design and programming have many parallels, students fear using unfamiliar representations; forcing them using pen-and-paper helped reduce that fear; does that work also for different program representations? Can we force students to get familiar with other representations?
- Michael: Even when using other representations, students are glued to code
- Amy: Start with input – output before submitting code and have students work out possible concrete algorithmic choices about how to process them, getting them use to algorithm design through low-fidelity representations.
- Kathi: Numberless word problems; give scenario and talk about; or take out question entirely and just talk about scenario; only then give out question; related to programming: what if we take out code and only talk about a problem?
- Tobias: How do I teach functions? switch between function and formula until they have understood
- Amy/Diane: representations of fractions for months until they really understood it; now they can apply that
- Tobias: Students do not see that 0.03 and 3% means the same

- Amy: in math, well-defined representations exist, but not in CS
- Michael: in CS representations, there are some variations in exactly what they mean; they aren't identical. That sometimes gets in our way.
- Ethel: Who decides what are equivalent representations?
- Kathi: What are the types of representations we want to see?

- Code
- Amy: Data Struct Transformation Comic Strip -- Show the state changes in a temporal form
- Tobias: Flow charts (Control flow)
- Michael: Data flow charts
- Memory-based representations (lists)
- Janet: user perspective (input/output)
- Recursion evaluation trees
- Kathi: All of these are very geeky -- what about the interaction with other parts of society?

- Amy: We should engage with design, but we can't teach all of it. We should do justice to it. Which parts do we pull off?
- Alannah Oleson, Amy J. Ko, Brett Wortzman (2020). On the Role of Design in K-12 Computing Education. ACM Transactions on Computing Education, Article 2. https://doi.org/10.1145/3427594
- Maybe they are only geeky in our conventional presentation, but could be brought in more richly through context/domains/etc

- Tobias: Tim Bell's 10 Principles of Computer Science
- Michael: We have useful tools but we need them to be higher level and to connect better.
- Amy: Maybe we need to use more relatable and relevant data.
- Amy: Need to not lose the meaning of data as we represent and work with it
- Tobias: Math edu: We need to formula problems as "word problems". Students need to then extract the information of the word problem before they solve the problem. We risk introducing additional requirements that students need to complete.
- Amy: But sometimes these requirements are skills we want to build.

5.4 Scaffolding Parts of the Process

Understanding the question
- Planning
  - Program Specification / Clarification
  - Program decomposition
  - Choosing overall model / data structures
- Translation – Data Model/Data Structure selection
- Implementation
- Testing
- Debugging
- Reflection

What would a data structure-inspired notational machine look like?
Highlight changes in data, not the control structure.

What variables should you print out when debugging?

Have a data-driven stepper – watch a subset of variables, and have the stepper highlight when each variable value changes. Updated watch variables done in a very friendly way.

Input-Output-Error does not mean the error is in the code, but maybe also on the conceptual level; can students identify relevant parts of source code? So move into a Parson's problem like setting

How to make reliable subgoal labels/plans that actually help students?

Give an example, and then let students work on a similar example to support their understanding

Strategies of students have limits; and if one strategy leads to a dead end, what to do? Allow them to continue that strategy? Highlight a different strategy?

When students have bugs, they often do not know what the intermediate steps are

Have a debugger that works with a student: Ask after each step of execution the state (e.g., fill in numbers in a linked list).

Students do not break down the problem (the debugging problem, not the coding problem), just focusing on the too big a problem, e.g., failing a test case; they do not do the detective work during debugging

Could we generate a sequence of questions to ask them that are the questions they should ask themselves when they debug?

Barb's grant:

Scaffold the TA-student interaction, so that TA can figure out what the misconceptions are;

Nested for loops vs iterating over two lists at the same time; many misconceptions, e.g., schema retrieval, workings of nested loops, incorrect model of data-program state; break down to first steps of loop?

How to detect student's misconceptions in their plans? Break down of iteration step by step, then go one step back

Soloway said that they understand control structures individually but do not understand how to compose them.

There are different kinds of misconceptions at different programming stages (and you would need different supports)

How do we scaffold going from algorithm to code?

Parson's problems of planning steps with vocabulary, e.g., order, clean, filter, aggregate

What kind of language would work for this process? Diana: data-driven approach to let students implement, then analyze and develop vocabulary that works for students

Experiment/study: Audience: starting students or upper level students? Does that make a difference? Also upper-level students have troubles planning

Ask students before and after implementation about their plans

Paper: Scaffolding design problems using Parson's problems: `https://dl.acm.org/doi/10.1145/3279720.3279746`

Should students implement standard data structures? What is the take away for students, what should they learn? Rather using data structures?

5.5 Sustainability notes

- Whatever the source of funding, it's key to align a project's fundraising strategy with the incentives, constraints, and expectations of the funding climate. For some, that might mean particular kinds of scholarship to justify sustain funding, for others that might mean adoption numbers, and for others still that might demonstrate impact on education

systems. Thinking carefully about these incentives is important, as they can often have novelty biases, rigor biases, and technology biases. Accounting for these biases, and ensuring they don't end up warping the scholarship, requires explicit planning.

- Another consideration is how much funding is restricted toward particular expenses; obviously, having unrestricted funding is the most valuable, as it allows for a project to meet whatever needs come, rather than being constrained. It's also the least common and hardest to obtain.
- There are almost always politics that influence how long money can be held and what it is spent on, whether it's a corporate politics game or an academic or foundation politics game. It's key to have someone who can manage those politics and relationships and ensure that they do not interfere with project goals in problematic ways. That could be a project lead, or a principal investigator, or even a funder or corporate partner who provides cover.
- There was a tangible sense of a continuum from deception to omission when it came to reporting and persuading funders. Everyone agreed that deception is unacceptable, but everyone also agreed that sometimes omission was necessary to amplify the outcomes that a funder might care most about, while hiding other details that might be in alignment with academic or innovation goals, but in tension with funder goals.
- It was quite common for successful projects to have one or two people with semi-permanent positions as the backbone of a project, from a maintenance perspective and from a resource perspective. This might be a corporate job or academic position. But it also means that projects have a single point of failure.
- Backend infrastructure can be a key risk to project sustainability. It requires regular maintenance, cloud costs, and staffing. Committing to back end infrastructure is a significant decision with long term consequences. Using university bandwidth and hosting is one way to avoid these costs, but is often restricted to static hosting, or has costs for university IT. But the quality of university IT service can be highly variable, and can also require some political negotiations to navigate policy restrictions.
- It's important to consider other sources of revenue. Communities can be a source of revenue. User events can have registration fees and that can generate unrestricted funding. But where to store that money can be complicated and impose accounting challenges. Donations can also be a source of revenue. Sometimes people will just give and this can also be a source of funding, especially when requests are targeted towards those with philanthropic capacity. Some talked about offering nearly meaningless premium services that offer almost nothing extra, but allow corporations that are often reluctant to donate to pay for a service. This might not sustain the project, but it can help sustain it. All of this requires a place to keep money, which may or may not be the backbone organization.
- We also talked about other sources of staffing, such as ways of trying to onboard, supervise, and engage students to contribute, for credit or for modest pay. There are all kinds of challenges of doing this, including ensuring they have sufficient expertise, that there is onboarding for them, and that they get feedback through code reviews or other mechanisms. We also talked about open source contributions and the limited value of "drive by" contributors that don't have enough context for the project to make meaningful contributions. Some talked about ways of engaging contributors socially first, to get context, and then have them contribute later after they have it.

5.6 How to Snap

Participants: **Jens, Elena, Michael Lee, Kenichi, Jadga, Mark**

5.6.1 Meta-Discussion

Snap is challenged to fit within two words:

- Some users come in from Scratch and know "sprite" and "costume"
- Some users come in from programming and think "objects" and "bitmaps"

It's hard to make all features visible when the programming is all visible.

- No API documentation
- Have to use drag feedback, like colors.
- Has to be an ecosystem – curriculum, communities, documentation.

- Current Snap manual is SICP-lite.

Remixing in Scratch – big part of the community culture.
New features in Snap support this sharing/remixing in more granular ways

- No real way to link code to people, because it's coming from SAP. There's a danger about a large corporate entity being able to track code and people.

Supporting teachers in sharing, remixing, and ownership in the community is even harder but more important than students.

- Hedy's support for teachers is fairly impressive.

Object model in Snap:

- Sprites and clones use Henry Lieberman style prototypes and delegation of slots. Can inherit something like y-position, so clones are linked vertically.

- Sprites are objects with methods and instance variables, and can send messages and data between each other.
- Can connect sprites so that they are sub-parts of one another.

- Scenes are totally different worlds.

6 Open problems

6.1 What studies should we do together? (and which not) / what collaborations could come out of this week?

Mark Proposed Study: Test transitions between our languages and tools. Pairs of us work on transitions from (for example) from Hedy to Pyret, or Snap to Hedy. (Requesting Ethel to be part of this!)-I love this Mark-Ethel.

This is Felienne: I love this too! Many of our Hedy classrooms do Scratch first! We even already have data of users of whether or not they have used Scratch before! We can totally do data analysis on behavioral differences between Hedy users with and without Scratch experience!

Of course, Mark is interested in ANYBODY transitioning from Teaspoon to ANY PL! I'm so interested to see how we support that transition.

Ethel-i think this is the right time to explore transitions, i would encourage people to participate in this and understand transfer at a deeper level. Also mapping constructs across different programming languages is something I'm looking forward to. The mappings can be put in a public domain later on if possible ->Love this idea!!

Teacher Transitions: Examine the transition/process of teachers' learning and choice of second language. Can I suggest you read Ethel's paper about this :) ->`https://dial.uclouvain.be/pr/boreal/object/boreal:251251`

- Ethel's paper is great <3 In our breakout, we also talked about the challenge of moving curriculum, learning content. Do teachers focus on making their content better, or transitioning to new languages and versions?
- Thanks Mark! I was trying to remember how we phrased it in our breakout. – Mike

Kathi wants to explore program representations/notional machines that center around data transformations, partly to help students create and execute debugging plans. What would such a representation look like? What differences might arise based on the semantics of the programming language? Would a data-centered approach help connect data to plans to programs? Johan: Don't you want more than debugging plans: also the kind of planning you discussed in your talk. Kathi: yes, plans as well. I'm wondering whether data-centered plans would also give debugging guidance, which would again tie to notional machines.

Johan proposed study: (Should probably be combined with others.) Different approaches to how we can support planning. Using Mio, Shriram's, and Kathi's approach to planning, add a testing based approach to it (as discussed in the break-out session this morning), and study how it works. Ben likes this idea. See related idea below. Youyou also likes this idea. Kathi is interested. Diana

Diana Proposed study: Cog Sci & strategies people pair up to better understand the processes involved in different steps of programming (program clarification, planning / decomposition, algorithm development, coding, debugging) and propose new strategies. Eva is keen to join :) Kathi is interested.

Ben: Program annotation, planning, and debugging – If students are writing interleaved (i.e. not very decomposed) programs, can they annotate pieces of their code to indicate which plans/goals pieces of their code are meant to accomplish? A visual representation of this could also be contrasted with a visualization of data flow within their programs (e.g. to indicate that some information (e.g. input from a test case) is not making it into a vital part of computing a correct result). This could be used to build more focused steppers or to help students to figure out where to focus when dealing with incorrect behavior. Johan: can we for example connect annotations/subgoal labels to test cases? Or ask for test cases at these annotations? Can we grey out parts of the program that are not part of a subgoal in a stepper/debugger? +Barb

Bastiaan: nice idea! Kathi is interested.

Diana: How can we provide real-time hints / scaffolding to help students design "good" test cases (black-box tests) for the purposes of both design and testing? +Barb

Kathi is also interested in this, noting that tests characterize a space of inputs/scenarios not just individual scenarios. I see this as a form of sensemaking.

Diana: Can we analyze problem statements and develop a process for identifying keywords or aspects of the problem statement that point towards specific aspects of the design and/or implementation? +Barb: This might also help students write good purpose statements.

Barb: Can we all study the same problem – like Kathi's shopping cart problem or the rainfall problem in a variety of contexts with a variety of scaffolding? +Johan: I think taking a particular example to discuss our ideas will be very fruitful +Tobias +Diana

- MIMN? Multi-institutional, multi-national?
- ITiCSE working group: "Everybody teach X in any way you want. We all use the same measure of learning/understanding at the end."

Kenichi: Programming language teachers in CS departments do not necessarily know how their ways of teaching are effective or not. It would be great if education experts can see the class (although it takes quite a long time) and make advice on how to improve the class and even how to conduct experiments. Conversely, CS people could hear what teachers want to teach and advise what is feasible technically.

Diana: Not quite a study, but it seems like a Computing Research Infrastructure NSF (USA) grant could fund the development of an open-source framework for testing many innovations (in one particular language) so that research could be pushed forward. This could help many people.

Amy proposal: I'd love to see us try 5-10 radically different approaches to educational programming languages than exist now, both to explore the design space more broadly through the lens of cultural responsiveness, but also to understand more deeply the role of syntax, semantics, and representation on learning, teaching, curriculum. (What would the world look like if we had 10 different mature Scratch/Snap etc platforms, but doing things very differently for very different audiences). Anyone want to write a $25 million 5 year grant? Kathi would be interested in brainstorming around or discussing this.

Ben would as well . . . depending upon how radical we can be :-)

References

1 Garcia, Dan and Ball, Michael and Garcia, Yuan. 2022. *Snap! 7 – Microworlds, Scenes, and Extensions!. Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 2. ACM: New York, NY USA.*

Participants

- Kenichi Asai
Ochanomizu University – Tokyo, JP
- Michael Ball
University of California – Berkeley, US
- Neil Brown
King's College London, GB
- Youyou Cong
Tokyo Institute of Technology, JP
- Barbara Ericson
University of Michigan – Ann Arbor, US
- Kathi Fisler
Brown University – Providence, US
- Diana Franklin
University of Chicago, US
- Elena Leah Glassman
Harvard University – Allston, US
- Mark J. Guzdial
University of Michigan – Ann Arbor, US
- Bastiaan Heeren
Open University – Heerlen, NL
- Felienne Hermans
Leiden University, NL
- Jadga Hügle
SAP SE – Walldorf, DE
- Johan Jeuring
Utrecht University, NL
- Amy Ko
University of Washington – Seattle, US
- Tobias Kohn
Utrecht University, NL
- Shriram Krishnamurthi
Brown University – Providence, US
- Michael J. Lee
NJIT – Newark, US
- Eva Marinus
Pädagogische Hochschule Schwyz, CH
- Jens Mönig
SAP SE – Walldorf, DE
- R. Benjamin Shapiro
University of Colorado – Boulder, US
- Janet Siegmund
TU Chemnitz, DE
- Ethel Tshukudu
University of Botswana – Gaborone, BW