



Introduction to Programming using Interactive Learning

1st Stephan Krusche 
Technical University of Munich
krusche@tum.de

2nd Jonnathan Berrezueta-Guzman 
Technical University of Munich
s.berrezueta@tum.de

Abstract—Interactive learning has been highly successful in computer science education, leading to improved planning and execution of programming courses. This is accomplished through the amalgamation of exercises equipped with real-time feedback mechanisms, fostering an environment conducive to iterative performance enhancement. Communication portals facilitate knowledge sharing between students and instructors, while computer-based exams alleviate the instructors' workload.

This paper focuses on the implementation of interactive learning in a programming course for first-year computer science students. The course, attended by an aggregate of 70 students, predominantly inexperienced in programming, is detailed, along with the digital resources employed during its development, implementation, and evaluation. This paper contributes to demonstrating the effectiveness of this methodology which showcase promising outcomes in a performance comparison between two courses with varying student numbers. The propitious outcomes emanating from this analysis undergird the proposition for extending this interactive learning methodology to a broader spectrum of computer science curricular offerings.

Index Terms—applied computing, learning management systems, education, learning management, learning success, online exams

I. INTRODUCTION

Programming necessitates the practical application of knowledge, and examples and exercises are vital for effective cognitive skill acquisition [1]–[3]. However, creating and evaluating programming exercises that promote cognitive skills and creativity can be time-consuming for instructors [4], [5]. Well-crafted examples contribute to better learning outcomes, and individual feedback through formative assessment is crucial for knowledge improvement [6], [7].

We have developed an interactive learning method to benefit students in the first semester of a computer science bachelor program, regardless of their prior programming knowledge. This method incorporates a tool with various functionalities to support both students and instructors in the learning and evaluation process.

The methodology includes in-class exercises to familiarize students with programming notation, group exercises in tutor sessions to practice and discuss advanced programming concepts, and homework exercises for independent skill development. Individual feedback on homework assignments enables students to assess their progress and enhance their programming skills.

This methodology promotes continuous feedback and problem-solving abilities alongside coding proficiency. Encouraged by positive outcomes, we implemented this methodology in a second-semester course with a large number of participants, achieving similarly promising results.

The paper is structured as follows: Section II covers related work, Section III describes the interactive learning method and tool support, Section IV presents a programming course that implemented the method, Section V discusses its contribution, and Section VI concludes the paper and suggests future work.

II. RELATED WORK

Programming is a dynamic and collaborative activity that requires logical thinking and identifying conceptual issues [8]. In education, it is crucial to develop engaging and interactive curricula [9]. Traditional approaches that separate content delivery from practice, such as lectures and exercises, often result in knowledge gaps. Ebbinghaus's forgetting curve suggests that rapid learning is followed by rapid forgetting [10]. It demonstrates that without practice, 40 % - 60 % of newly acquired knowledge can be lost in the first 24 hours [11].

Pedagogical concepts like *active learning*, *experiential learning*, *blended learning*, and *interactive learning* have emerged to bridge the gap between content delivery and practice [12]–[15]. Involving students in the learning process, these approaches enhance information retention through engagement, instructor observation, and critical reflection [10]. One such approach is Think-Pair-Share (TPS), where students tackle a problem individually, then collaborate in small groups, and ultimately share their ideas with the entire class [16].

Kothiyal and his colleagues conducted a study in a large introductory programming course. They observed the student's engagement patterns during 10 weeks and 13 TPS activities [17]. A 83 % students' engagement rate was reported in the end. The study [18] proposed the concept of interactive lectures, that integrates teaching and exercises in short iterations. This approach achieved a student engagement rate of 80 %.

Robinson and Carroll focus on the development and deployment of an open-source online learning platform designed for teaching and evaluating programming [19]. The results indicated that students perceive real-time feedback as an effective means of experimentation and self-directed learning. From the perspective of the instructors, workload and required effort for evaluation are reduced significantly.

David Shaffer introduced a platform designed for grading and providing feedback during programming assignments [2]. Students can improve their programming skills, while instructors can create assignments that can be automatically graded.

Wang and Liang introduce an online system developed for class management, assignment and exam creation, and tracking class performance [20]. It enables instructors to assess the overall class performance and identify individual student difficulties. It provides detailed feedback for each submission, allowing students to debug their code also. In the case of syntax errors, the feedback includes comprehensive compilation error messages and syntax hints. Likewise, for logic errors, the feedback presents an example of the error.

Chaordic learning is as a self-organizing, adaptive, and non-linear pedagogical methodology, primarily devised to catalyze the innovative cognitive processes in learners [21]. Educators employ an organized scaffold, extending mentorship and direction, while concurrently incorporating a certain degree of liberty to promote self-regulation and autonomous learning. This educational approach thereby fosters an environment conducive to the cultivation of innovation and creative thinking.

III. LEARNING METHOD

Interactive learning is an educational approach centered around student interaction and based on the constructive alignment philosophy [22].

This methodology minimizes the gap between teaching new concepts and their practical application through active student participation in the classroom. It incorporates creativity to foster problem-solving and soft skills. Instructors engage in short cycles of teaching and exercising concepts, providing immediate and personalized feedback using tool support [23]. Each cycle comprises five activities within a lecture:

- 1) **Theory:** Introduce and explain a new concept.
- 2) **Example:** A concrete example to illustrate the concept.
- 3) **Practice:** Apply the new concept in an exercise.
- 4) **Feedback:** On students' exercise submissions.
- 5) **Reflection:** Foster discussion among students.

A. Deepening the Learning Content

Interactive learning encompasses various types of exercises (such as modeling exercises) beyond in-class activities [18], [24]. Group exercises are conducted in smaller tutor sessions, providing students with the opportunity to apply and practice the taught concepts alongside their peers. On the other hand, homework exercises with individual feedback enable students to gauge their learning progress and refine their skills. The presentation of solutions for group and homework exercises in tutor sessions also enhances students' communication skills.

B. Exercises

The learning method includes three types of exercises.

1. *Short in-class exercise:* During the lecture, students engage in solving brief exercises that prompt the application of the reviewed knowledge. Students have 5-10 minutes to attempt the exercises independently. After this time, the instructor addresses any doubts and provides further clarification.

2. *Group exercise:* During the tutor sessions, students tackle two medium-difficult exercises that offer immediate feedback. Their solutions are presented by the students in the session, allowing for explanation and discussion with their peers and tutor. This collaborative approach helps solidify the understanding of the concepts and prepares them for more challenging exercises, such as homework.

3. *Homework exercise:* These exercises are designed to be challenging, requiring students to independently apply the knowledge gained from the lecture. These exercises typically provide immediate feedback to facilitate continuous learning. After the deadline, the solutions to the homework exercises are presented in tutor sessions, allowing for in-depth discussions and the resolution of any doubts.

C. Tool Support

Artemis is an open-source exercise system¹ that incorporates features such as distributed version control and continuous integration to support the automatic assessment of programming exercises [25]. This enables efficient and streamlined evaluation of student work in programming exercises.

Instructors can create exercises writing a problem statement in Markdown in the web interface and configure three git repositories: the template with starter code, the solution with the correct implementation, and the test repository with the tests used for auto-assessment and the individual feedback. Additionally, Artemis sets up two build plans, one for the template (to ensure it is assessed with 0 %) and one for the solution (to ensure it is assessed with 100 %). One build plan checks out the template (or solution) repository and the test repository and executes all tests against the template, solution, or student code. This process is better explained in Figure 1.

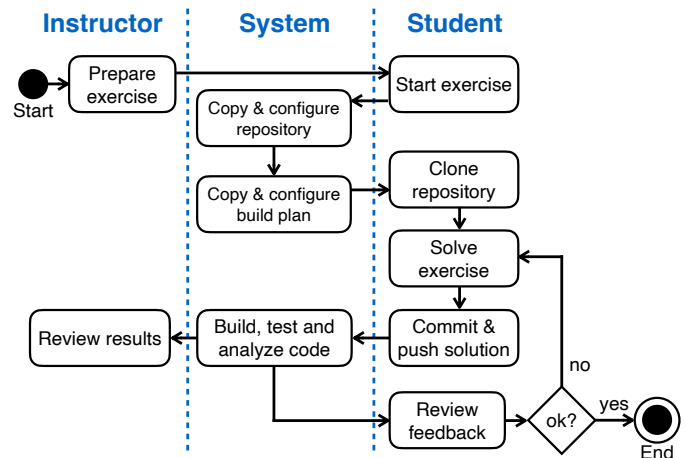


Fig. 1. Simplified activity diagram for conducting programming exercises with Artemis [25].

Students can see the problem statement and clone the template repository and the template build plan. Each student receives one individual repository and one build plan per exercise. They can solve the exercise in their local IDE (e.g.

¹<https://github.com/lslintum/Artemis>

IntelliJ) and then commit and push their solution. This triggers their individual build plan to compile, build, test, and analyze their code and provide feedback based on the instructor’s tests. Students can review this individual feedback immediately and adapt their solutions accordingly. Instructors optionally can activate and configure static code analysis.

Artemis includes interactive and dynamic exercise instructions [25]. Figure 2 shows a screenshot of a problem statement where correctly implemented tasks are marked in green otherwise in red. This helps students to identify which parts of the exercise they have already solved correctly and improves the understanding of the source code.

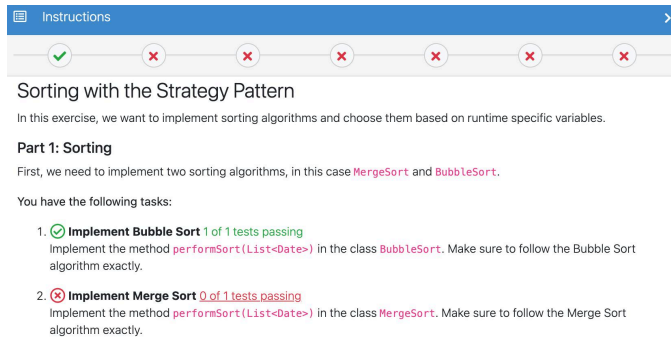


Fig. 2. User interface that shows a problem statement of an exercise with the interactive feedback. It provides students with feedback based on test results and static code analysis. Green indicates positive feedback, red indicates unresolved tasks.

D. Assessment

Artemis provides 3 assessment types for exercises.

1. *Automatic*: The instructor writes test cases that run for each submission either during (for in-class, group, and homework exercises) or after the deadline (exam exercises).

2. *Semi-automatic*: Some implementations are partially impossible to test automatically, e.g. graphical user interface (GUI). In this case, the GUI part of the exercise should be deployed and evaluated manually by the instructor.

3. *Manual*: When a programming exercise cannot be evaluated automatically, the instructors download the repositories and review or debug the code submitted by students. This evaluation process is double-blind, neither the student nor the reviewer knows each other.

Students can submit a complaint and a second reviewer answers and modifies the grade if necessary. Students can also submit "more feedback requests" and the same reviewer will provide an extended explanation without editing the grade.

IV. COURSE

The combined first semester course **InProg** (In: Introduction to Informatics, and Prog: Fundamentals of Programming) is a mandatory courses in the Bachelor of Information Engineering². It introduces students to programming using the

interactive learning method described in Section III. InProg consists of 4 hours of lecture and 4 hours of tutor group exercises per week. The learning objectives are that students get acquainted with object-oriented and functional programming. Assignments apply problem-solving concepts such as basic data structures, recursion, classes and methods, lists, GUI creation, concurrency, and advanced object-oriented programming concepts. The final grade is determined by the computer-based final exam (for In) and the score of weekly homework exercises (for Prog).

The course uses constructive alignment to align teaching and assessment with the course objectives [26]. Each lecture defines a set of learning goals based on the six cognitive skills in Bloom’s taxonomy [27]. The instructors focus on higher cognitive skills so that students learn to apply concepts in concrete situations.

A. Organization

One lecturer and one exercise instructor organized and conducted the course with the help of 4 student tutors (students who successfully approved before). The course took place in the winter semester (2021-2022) over 13 weeks. In total, 70 students were registered for the course, and around 60% of them participated actively in the lecture hall while the students who attended the streamed lecture could interact using Zulip³.

To deal with the challenge of keeping students motivated throughout the semester, the course includes interactive elements such as in-class quizzes with multiple-choice, drag-and-drop, or short-answer questions to keep them engaged throughout the course. The quizzes help to recapture learned content and their questions are focused on problem-solving.

B. Programming Exercises

Students learn to make connections and see differences between models and their implementation in programming exercises. This stimulates their cognitive skills and they learn to apply the knowledge when implementing source code.

For the creation of a programming exercise, problems with a possible real context are proposed, avoiding the use of typical variables (x , y , etc.). In this way, students take ownership of the problem. Students are provided with a repository with a template for developing the solution which has TODOs comments to guide students on where and how they should provide their solution. The tests are focused on providing immediate feedback to students allowing them to iterate their solution until completing the tests that evaluate the exercise.

Artemis allows students to post questions about an exercise. Instructors, tutors, and students can react to the questions and answer them. Additionally, instructors can post announcements that are accompanied by an email to the students.

Programming exercises are mostly automatically evaluated. The feedback that students receive about their submissions is crucial to understand how they can improve. In the following,

²Information Engineering is a study program comparable to Computer Science which integrates additional management and engineering aspects.

³Zulip is an open-source instant messaging platform that can easily be hosted by a university. Link: zulip.com

we describe examples of exercises used in week 11 for teaching the topic of GUIs.

Short in-class exercise: L11E03 Email Generator: Students develop a GUI for creating an email address using two text fields, one button, and a label. This exercise put into practice the use of controls and layouts with JAVA FX. The estimated time to solve this exercise is 10 minutes, and when the time is up, the lecturer solves it and explains the solution.

Group exercise: G11E01 Airport Check-In: Students develop a GUI for an airway check-in portal. They have to implement controls, different layouts, usability principles, and some logic around the functions of these controls. The estimated time to solve this exercise in the tutor session is approx 45 minutes.

Homework exercise: H11E01 ToDo List: Students implement a Task-List with a GUI that creates, deletes, sorts, and updates task items or marks them as solved. In addition, students apply Nielsen’s 10 heuristics principles taught in the lecture [28]. This is an exercise that allows the student to be creative as well. The implementation of the logic part is evaluated automatically while the GUI implementation part is evaluated manually (semi-automatic assessment). The estimated time to solve this exercise is 120 minutes.

C. Tutor Exercise Sessions

Four tutors hold tutor sessions weekly with around 15 students to moderate discussions and review the learned concepts in case the students have questions. Students apply the knowledge acquired in the lecture solving the group exercises. Each tutor exercise session is structured as follows:

- 1) **Review of previous lecture** [5 - 10 min]: students discuss the learning goals, outline, and summary.
- 2) **Homework presentation** [20 - 30 min]: students present their solution to homework exercises.
- 3) **Group work** [90 - 120 min]: Students work on predefined group exercises in groups (3-6 students).
- 4) **Discussion of next homework** [5 min]: the new homework exercises are briefly discussed.

D. Final Exam

The final evaluation for InProg is an on-site computer-based exam, which is composed of two quiz exercises, one modeling exercise, and two programming exercises representing 90 points in total. Each exercise has multiple variants for difficult cheating. Two weeks before the students get familiar with the exam mode in Artemis with a test exam.

The exam follows a constructive alignment and assesses the competencies taught in the learning activities of the course that are aligned with the learning goals. The quiz exercises include 10 questions. The modeling exercise presents a code and asks to provide the corresponding flow chart diagram. The first programming exercise is focused on object-orientated programming and streams (automatic assessment). The second programming exercise is focused on GUIs and testing (semi-automatic assessment). Students do not retrieve feedback during the exam but only see if their code compiles or not.

For the exam, 48 students were registered (68.5 % of the registered students in the course), 44 of them started the exam, and 41 submitted a solution. In the end, 25 students (56 % of the submitted ones) passed the exam.

E. Scaling the Interactive Learning Methodology

When the InProg course concluded, 62 students from this group participated in the following course in the second semester, Introduction to Software Engineering (**ISE**)⁴ which introduces different software engineering concepts and evolves the development of programming exercises. This course applied the *interactive learning* methodology and was also conducted on another campus in another city with a group of 2,163 students. The participation and performance of both groups were similar (see Table I). Group 1 includes students who participated in InProg and Group 2 includes students from the other city with the same course.

TABLE I
OBTAINED RESULTS OF APPLYING INTERACTIVE LEARNING IN COURSES WITH A DIFFERENT NUMBER OF PARTICIPANTS.

Parameter	Group 1	Group 2
Registered students in the course	62 (100 %)	2,163 (100 %)
Exam participation	44 (71 %)	1,646 (76 %)
Approved students	25 (56 %)	1,055 (64 %)

Finding 1: Initial exposure and subsequent application of interactive learning, across groups of varying sizes, lead to improved outcomes that show scalability for education, even in courses with large enrollments.

V. DISCUSSION

The *Interactive learning* methodology has significantly enhanced programming education and problem-solving through coding. By offering contextual programming exercises, students are incentivized to meticulously analyze and iterate over solutions. Continuous feedback bolsters programming proficiency and resilience.

Collaborative dynamics between students and instructors via communication portals enable multifaceted solution discussions. In-class programming exercises fortify concepts, while quizzes foster clarifying discussions on prior topics.

The participation and performance of the students are shown in Figure 3. The performance during the first two weeks is high because the basic concepts were presented. However, the performance in week 10 is low due to the complexity of the content (threads). This performance improved in the following three weeks because the students were able to earn bonus points with optional challenges.

The utilization of computer-based examinations streamlines the assessment procedure, substantially mitigating the expenditure of instructor effort.

The student evaluation of the course presented many positive results. Students liked that they received immediate and

⁴It is not required to pass InProg for ISE but it still highly recommended.

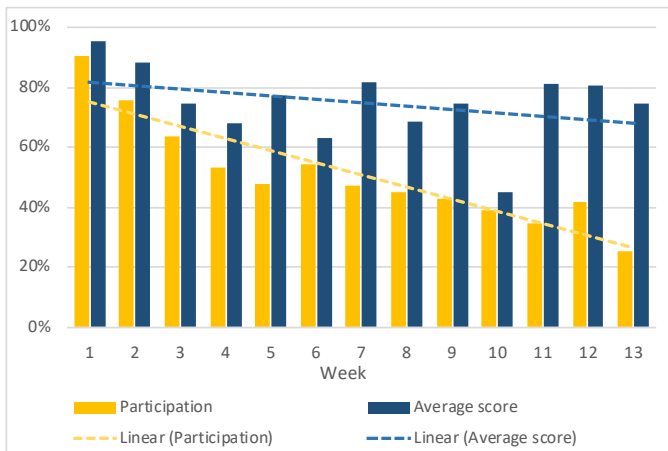


Fig. 3. Average participation and the average score in the Homework exercises per week during the InProg course.

helpful feedback for their exercise submissions. One student e.g. stated: “The course is very well structured in the way that there are incrementally difficult exercises which allow us to ease into the topics”.

Finding 2: Students emphasize the benefits of tutorials to improve understanding of taught concepts and to prepare for homework assignments.

VI. CONCLUSION

Interactive learning simplifies the conduction of programming exercises by integrating distributed version control and continuous integration. This pedagogical approach equips students with the tools and resources to learn through automated feedback in group and individual exercises. Additionally, it fosters interaction in scheduled tutorial sessions and facilitates the execution of assessments on personal computing devices.

Notwithstanding the encouraging outcomes associated with this methodology, it has become apparent that emergent challenges warrant attention in subsequent iterations. A particular area of concern is the quantification of the time investment required by each individual exercise. Consequently, an adaptive approach that calibrates the complexity of exercises in response to the diverse competencies of students in heterogeneous groups is projected as a significant enhancement.

This paper substantiates the efficacy of interactive learning as a viable instructional strategy in introductory programming courses, delineating its advantages for both learners and educators. The insights pertaining to the benefits and challenges discussed herein are envisaged to inform the adoption and adaptation of this methodology across various educational domains.

REFERENCES

[1] T. Connolly, M. Stansfield, and T. Hainey, “An application of games-based learning within software engineering,” *British Journal of Educational Technology*, vol. 38, no. 3, pp. 416–428, 2007.

[2] D. Shaffer, “Pedagogical praxis: The professions as models for post-industrial education,” *Teachers College Record*, vol. 106, no. 7, pp. 1401–1421, 2004.

[3] K. VanLehn, “Cognitive skill acquisition,” *Annual Review of Psychology*, vol. 47, pp. 513–539, 1996.

[4] J. Sweller and G. Cooper, “The use of worked examples as a substitute for problem solving in learning algebra,” *Cognition and Instruction*, vol. 2, no. 1, pp. 59–89, 1985.

[5] J. G. Trafton and B. Reiser, “Studying examples and solving problems: Contributions to skill acquisition,” tech. rep., Naval HCI Research Lab, Washington, DC, USA, 1993.

[6] R. Higgins, P. Hartley, and A. Skelton, “The conscientious consumer: Reconsidering the role of assessment feedback in student learning,” *Studies in higher education*, vol. 27, no. 1, pp. 53–64, 2002.

[7] A. Irons, *Enhancing learning through formative assessment and feedback*. Routledge, 2007.

[8] J. Whitehead, “Collaboration in software engineering: A roadmap,” *FOSE*, vol. 7, pp. 214–225, 2007.

[9] K. Livingstone, “Constructive alignment and the curriculum: a call for improved pedagogical practices in higher education,” *Blue Ocean Research Journals*, vol. 3, no. 12, pp. 19–34, 2014.

[10] H. Ebbinghaus, “Memory: A contribution to experimental psychology,” *Annals of neurosciences*, vol. 20, no. 4, p. 155, 2013.

[11] J. Murre and J. Dros, “Replication and analysis of ebbinghaus’ forgetting curve,” *PloS one*, vol. 10, no. 7, 2015.

[12] C. Bonk and C. Graham, *The handbook of blended learning: Global perspectives, local designs*. John Wiley & Sons, 2012.

[13] C. Bonwell and J. Eison, *Active Learning: Creating Excitement in the Classroom*. ASHE-ERIC Higher Education Reports., 1991.

[14] D. Buehl, *Classroom strategies for interactive learning*. Stenhouse Publishers, 2017.

[15] A. Kolb and D. Kolb, “Learning styles and learning spaces: Enhancing experiential learning in higher education,” *Academy of management learning & education*, vol. 4, no. 2, pp. 193–212, 2005.

[16] F. Lyman, “Think-pair-share: An expanding teaching technique,” *Maa-Cie Cooperative News*, vol. 1, no. 1, pp. 1–2, 1987.

[17] A. Kothiyal, R. Majumdar, S. Murthy, and S. Iyer, “Effect of think-pair-share in a large cs1 class: 83% sustained engagement,” in *Proceedings of the 9th annual international conference on International computing education research*, pp. 137–144, ACM, 2013.

[18] S. Krusche and A. Seitz, “Increasing the interactivity in software engineering moocs - A case study,” in *52nd Hawaii International Conference on System Sciences*, pp. 1–10, 2019.

[19] P. Robinson and J. Carroll, “An online learning platform for teaching, learning, and assessment of programming,” in *Global Engineering Education Conference*, pp. 547–556, IEEE, 2017.

[20] J.-Y. Wang and J.-C. Liang, “Codinghere: Online judge and assessment system for programming course,” in *5th Eurasian Conference on Educational Innovation*, pp. 126–129, IEEE, 2022.

[21] S. Krusche, B. Bruegge, I. Camilleri, K. Krinkin, A. Seitz, and C. Wöbker, “Chaordic Learning: A Case Study,” in *39th International Conference on Software Engineering: Software Engineering Education and Training*, pp. 87–96, IEEE, 2017.

[22] G. Dames, “Enhancing of teaching and learning through constructive alignment,” *Acta Theologica*, vol. 32, no. 2, pp. 35–53, 2012.

[23] S. Krusche, N. von Frankenberg, and S. Afifi, “Experiences of a software engineering course based on interactive learning,” in *15. Workshops Software Engineering im Unterricht der Hochschulen*, pp. 32–40, 2017.

[24] S. Krusche, N. von Frankenberg, L. M. Reimer, and B. Bruegge, “An interactive learning method to engage students in modeling,” in *42nd International Conference on Software Engineering, Software Engineering Education and Training*, pp. 12–22, ACM, 2020.

[25] S. Krusche and A. Seitz, “ArTEMiS: An Automatic Assessment Management System for Interactive Learning,” in *49th Technical Symposium on Computer Science Education*, pp. 284–289, ACM, 2018.

[26] J. Biggs, “Aligning teaching and assessing to course objectives,” *Teaching and learning in higher education: New trends and innovations*, vol. 2, pp. 13–17, 2003.

[27] B. Bloom, M. Engelhart, E. Furst, W. Hill, and D. Krathwohl, “Taxonomy of educational objectives: The classification of educational goals,” 1956.

[28] J. Nielsen, “Enhancing the explanatory power of usability heuristics,” in *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pp. 152–158, 1994.