Recap SWE233

SWE233: Intelligent User Interfaces

https://dayenam.com/teaching/swe233-fall2025/

Daye Nam

Fall 2025

Review of IUI papers

We read 20 papers!

-2010: 5
2011-2022: 6
2023-:9

ICSE/FSE: 5
CHI: 3
UIST: 7
Programming Language: 2
Others (Arxiv, IUI, ITS): 3

Oct 13	Integrated Development Environment (IDE) • *Code bubbles: rethinking the user interface paradigm of integrated development environments* • *Productivity Assessment of Neural Code Completion*
Oct 15	Conversational UI I • *Grounded Copilot: How Programmers Interact with Code-Generating Models* • *Why Jonny Can't Prompt: How Non-AI Experts Try (and Fail) to Design LLM Prompts*
Oct 20	Conversational UI II • *Context-aware conversational developer assistants* • *Need Help? Designing Proactive AI Assistants for Programming*
Oct 22	Agents • *Magentic-UI: Towards Human-in-the-loop Agentic Systems* • *Generative Agents: Interactive Simulacra of Human Behavior*
Oct 27 *Due: Presentation slides*	Project Proposal & Feedback 1
Oct 29	Project Proposal & Feedback 2
Nov 3 *Due: Project Proposal*	User Modeling & Personalization • *Creating General User Models from Computer Use* • *Supporting Reuse by Delivering Task-Relevant and Personalized Information*
Nov 5	End Users I • *SQLucid: Grounding Natural Language Database Queries with Interactive Explanations* • *Situated Live Programming for Human-Robot Collaboration*
Nov 10	End Users II • *ProgramAlly: Creating Custom Visual Access Programs via Multi-Modal End-User Programming* • *Small-Step Live Programming by Example*
Nov 12	UI • *Auto-Icon: An Automated Code Generation Tool for Icon Designs Assisting in UI Development* • *Sikuli: Using GUI Screenshots for Search and Automation*
Nov 17 *Due: Prototype Design*	Collaboration • *Code space: touch + air gesture hybrid interactions for supporting developer meetings* • *What would other programmers do: suggesting solutions to error messages*
Nov 19	Debugging & Testing • *Debugging reinvented: asking and answering why and why not questions about program behavior* • *NaNofuzz: A Usable Tool for Automatic Test Generation*

Code Bubbles: Rethinking the User Interface Paradigm of Integrated Development Environments

IDE

Andrew Bragdon[1], Steven P. Reiss[1], Robert Zeleznik[1], Suman Karumuri[1], William Cheung[1], Joshua Kaplan[1], Christopher Coleman[1], Ferdi Adeputra[1], Joseph J. LaViola Jr.[2]

[1]Brown University
Department of Computer Science
{acb, spr, bcz, suman, jak2, wcheung, cjc3, fadeputr}@cs.brown.edu

[2]University of Central Florida
School of EECS
jjl@eecs.ucf.edu

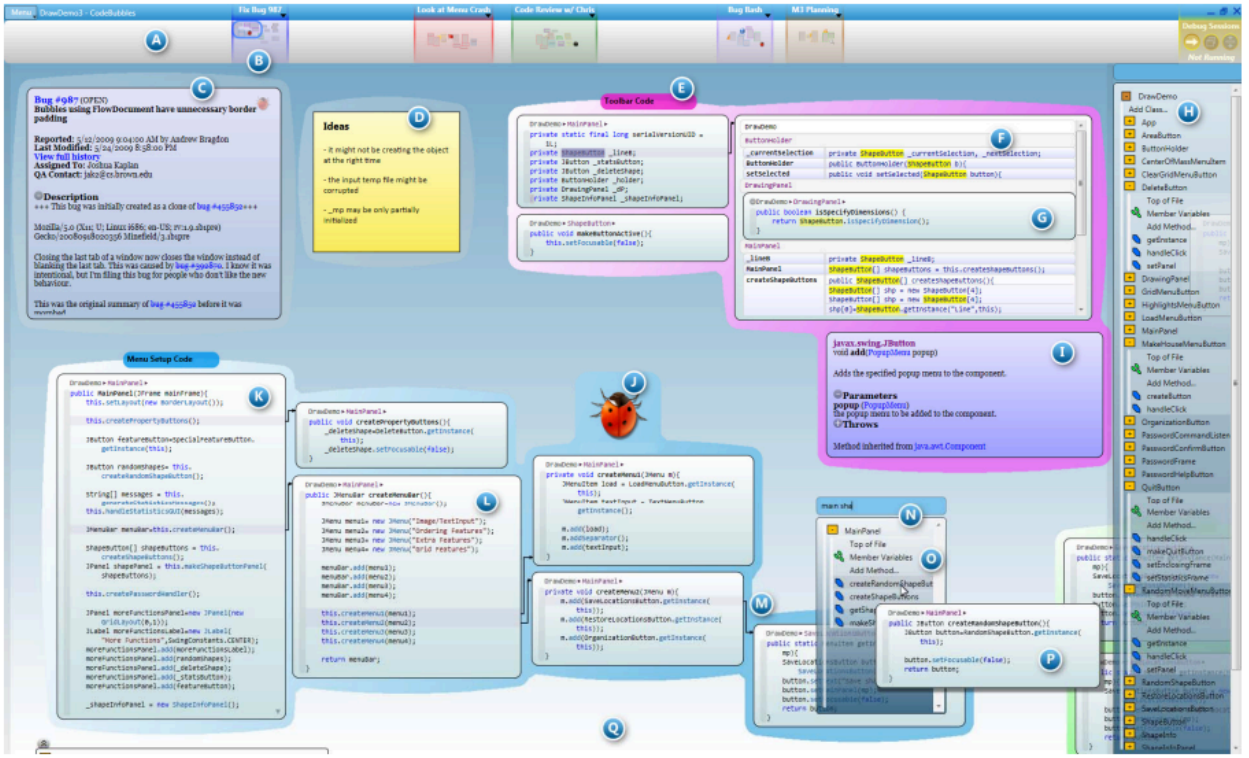
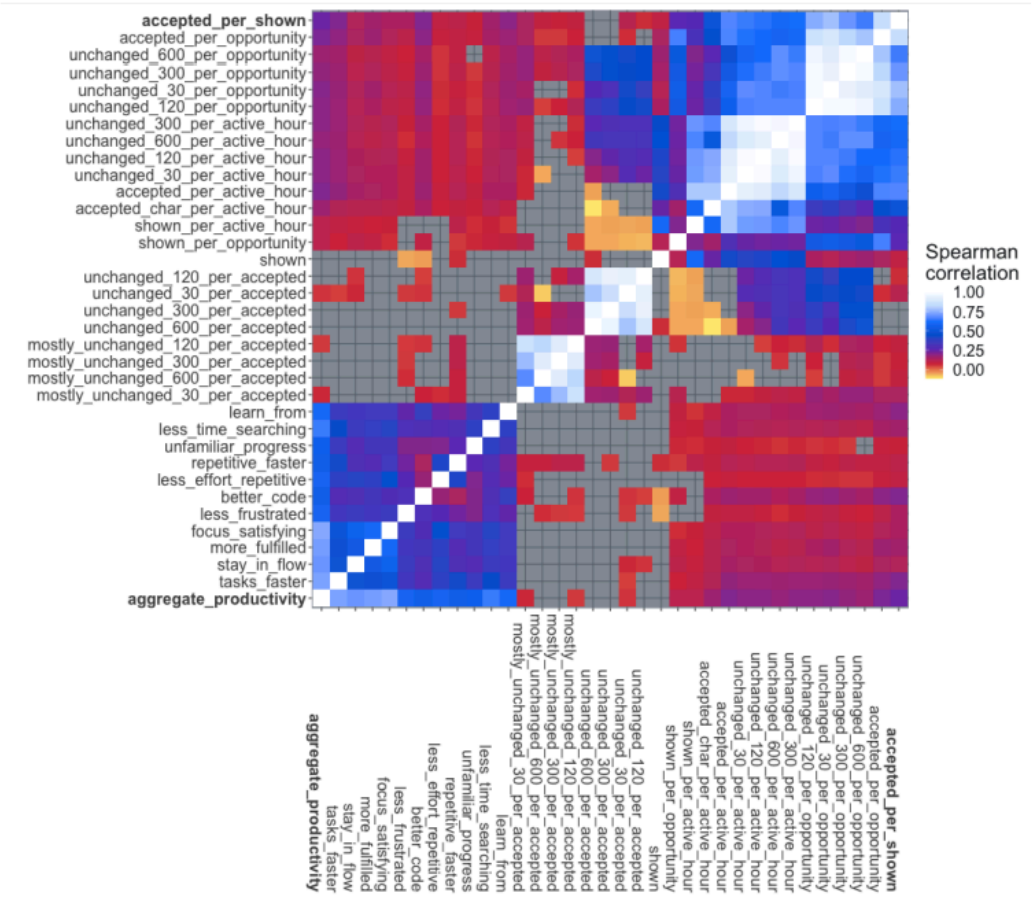


Figure 1. *The Code Bubbles IDE. See section 2 below, Scenario, and section 6 below, IDE User Interface, below. Resolution: 1920x1200 (space reserved for taskbar).*

Productivity Assessment of Neural Code Completion

Albert Ziegler, Eirini Kalliamvakou, Shawn Simister, Ganesh Sittampalam, Alice Li, Andrew Rice,
Devon Rifkin, and Edward Aftandilian
{wunderalbert,ikaliam,narphorium,hsenag,xalili,acr31,drifkin,eaftan}@github.com
GitHub, Inc.
USA

IDE



Grounded Copilot: How Programmers Interact with Code-Generating Models

SHRADDHA BARKE*, UC San Diego, USA
MICHAEL B. JAMES*, UC San Diego, USA
NADIA POLIKARPOVA, UC San Diego, USA

Conversational UI

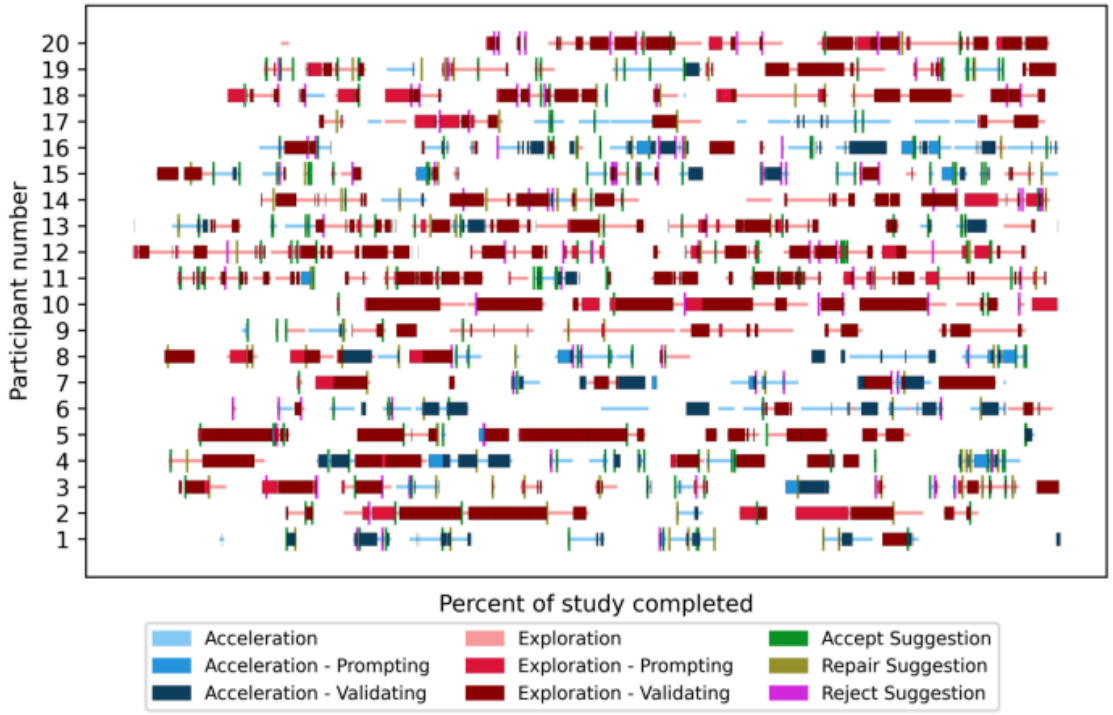


Fig. 3. Timeline of observed activities in each interaction mode for the 20 study participants. The qualitative codes include different prompting strategies, validation strategies and outcomes of Copilot's suggestions (accept, reject or repair)

Conversational UI

Why Johnny Can't Prompt: How Non-AI Experts Try (and Fail) to Design LLM Prompts

J.D. Zamfirescu-Pereira
zamfi@berkeley.edu
UC Berkeley
Berkeley, CA, USA

Richmond Wong
rwong34@gatech.edu
Georgia Institute of Technology
Atlanta, Georgia, USA

Bjoern Hartmann
bjoern@eecs.berkeley.edu
UC Berkeley
Berkeley, CA, USA

Qian Yang
qianyang@cornell.edu
Cornell University
Ithaca, NY, USA

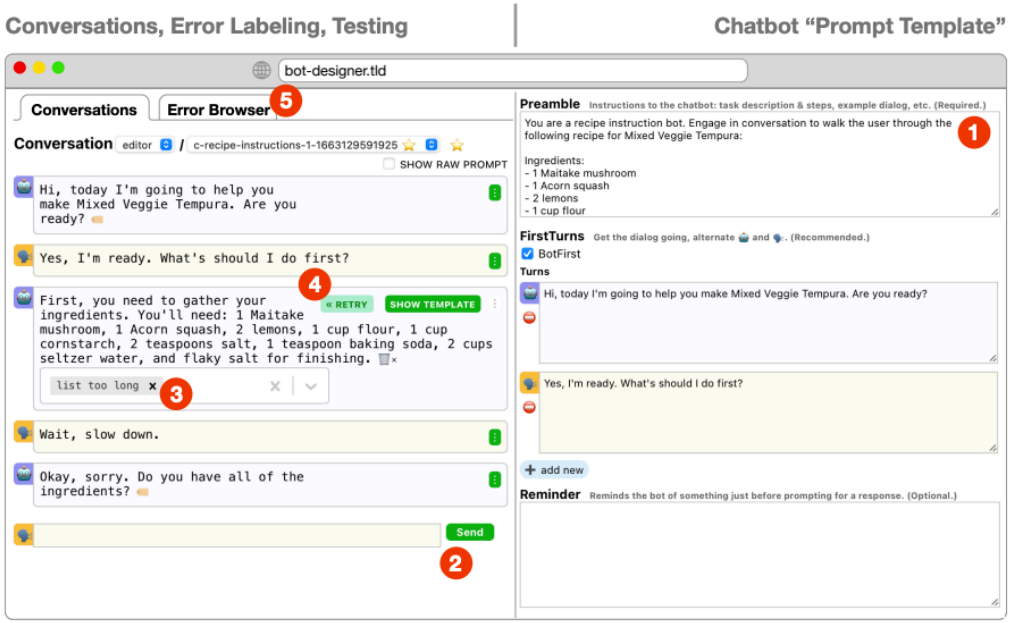


Figure 1: BotDesigner main user interface. Right pane: the *prompt template* authoring area (1); designers use plain text here to describe desired behavior for the chatbot they are developing. Left, tabbed pane: the conversation authoring area; here, designers can send chat messages (2) to the bot defined by the current template on the right. A *label* button reveals a labeling widget (3) that allows designers to attach an arbitrary text label to any bot-produced chat message, for aggregation and later identification. A *retry* button (4), only visible after the *prompt template* has been changed, lets designers test what new bot chat message would be produced, at that specific point in the conversation, as a result of the prompt template change. The "Error Browser" tab (5) reveals a UI panel for designers to test prompt changes against all labeled bot responses, across all conversations (see Figure 2).

Context-Aware Conversational Developer Assistants

Conversational UI

Nick C. Bradley
Department of Computer Science
University of British Columbia
Vancouver, Canada
ncbrad@cs.ubc.ca

Thomas Fritz
Department of Informatics
University of Zurich
Zurich, Switzerland
fritz@ifi.uzh.ch

Reid Holmes
Department of Computer Science
University of British Columbia
Vancouver, Canada
rtholmes@cs.ubc.ca

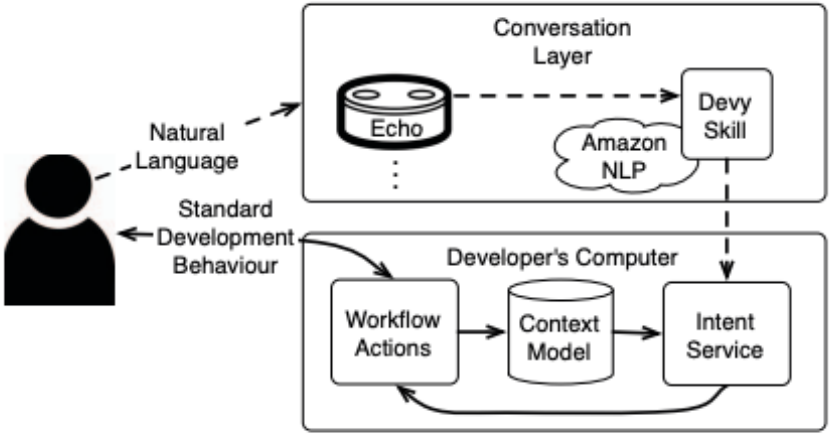


Figure 1: Devy's architecture. A developer expresses their intention in natural language via the conversational layer. The intent service translates high-level language tokens into low-level concrete workflows which can then be automatically executed for the developer. Dotted edges predominantly communicate in the direction of the arrow, but can have back edges in case clarification is needed from the user.

Need Help? Designing Proactive AI Assistants for Programming

Valerie Chen
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
valeriechen@cmu.edu

Alan Zhu
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
alanzhuyixuan@gmail.com

Sebastian Zhao
University of California Berkeley
Berkeley, California, USA
sebbyzhao@berkeley.edu

Hussein Mozannar
Microsoft Research
Redmond, Washington, USA
hmozannar@microsoft.com

David Sontag
Massachusetts Institute of Technology
Boston, Massachusetts, USA
dsontag@mit.edu

Ameet Talwalkar
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
atalwalk@andrew.cmu.edu

Conversational UI

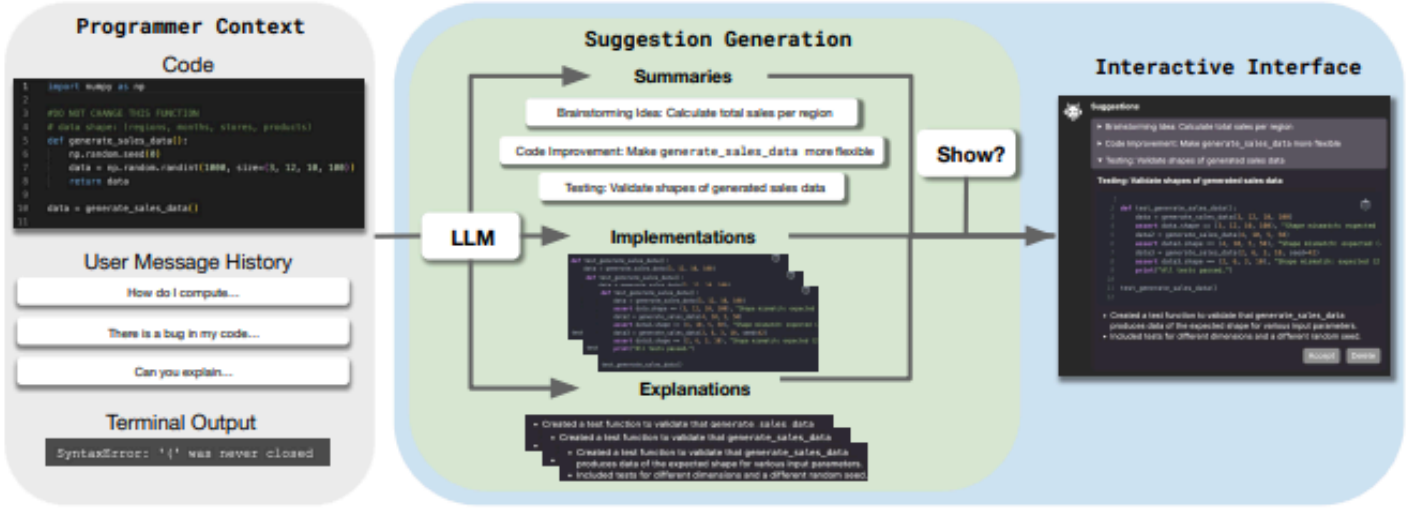


Figure 1: The implementation of a proactive chat assistant for programming. We introduce a proactive assistant that takes in the programmer's context, which includes the current code, user message history, and optionally terminal output, generates a set of suggestions, which include a summary, implementation, and explanation of implementation, and then determines whether it is timely to show the user in the interactive interface.

Agents

Magentic-UI: Towards Human-in-the-loop Agentic Systems

Hussein Mozannar, Gagan Bansal, Cheng Tan, Adam Fourney, Victor Dibia, Jingya Chen, Jack Gerrits, Tyler Payne, Matheus Kunzler Maldaner, Madeleine Grunde-McLaughlin, Eric Zhu, Griffin Bassman, Jacob Alber, Peter Chang, Ricky Loynd, Friederike Niedtner, Ece Kamar, Maya Murad, Rafah Hosn, Saleema Amershi

Microsoft Research AI Frontiers

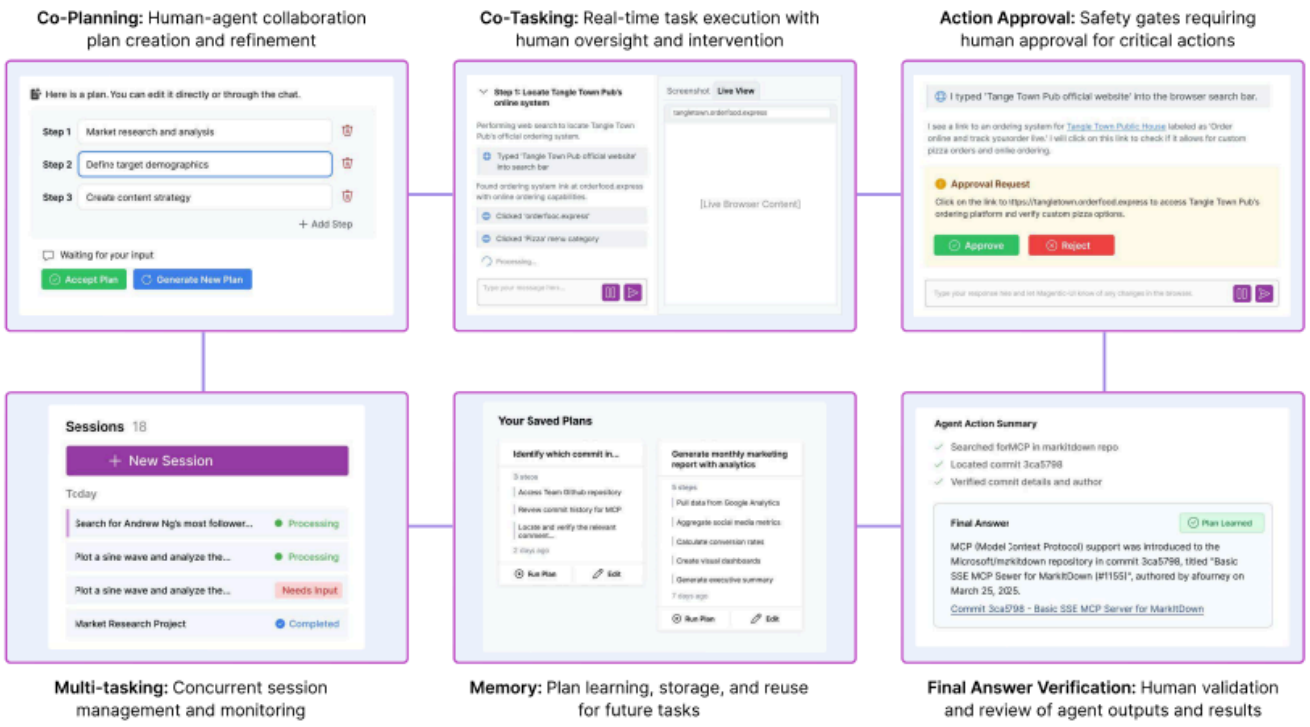


Figure 1: Magentic-UI is an open-source research prototype of a human-centered agent that is meant to help researchers study open questions on human-in-the-loop approaches and oversight mechanisms for AI agents.

Agents

Generative Agents: Interactive Simulacra of Human Behavior

Joon Sung Park
Stanford University
Stanford, USA
joonspk@stanford.edu

Joseph C. O'Brien
Stanford University
Stanford, USA
jobrien3@stanford.edu

Carrie J. Cai
Google Research
Mountain View, CA, USA
cjcai@google.com

Meredith Ringel Morris
Google DeepMind
Seattle, WA, USA
merrie@google.com

Percy Liang
Stanford University
Stanford, USA
pliang@cs.stanford.edu

Michael S. Bernstein
Stanford University
Stanford, USA
msb@cs.stanford.edu

Figure 1: Generative agents are believable simulacra of human behavior for interactive applications. In this work, we demonstrate generative agents by populating a sandbox environment, reminiscent of The Sims, with twenty-five agents. Users can observe and intervene as agents plan their days, share news, form relationships, and coordinate group activities.

User Modeling

Creating General User Models from Computer Use

Omar Shaikh
Stanford University
Stanford, CA, USA
oshaikh@stanford.edu

Shardul Sapkota
Stanford University
Stanford, CA, USA
sapkota@stanford.edu

Shan Rizvi
Independent
New York, NY, USA
shanrizvi4@gmail.com

Eric Horvitz
Microsoft
Redmond, WA, USA
horvitz@microsoft.com

Joon Sung Park
Stanford University
Palo Alto, CA, USA
joonspk@stanford.edu

Diyi Yang
Stanford University
Stanford, CA, USA
diyiy@cs.stanford.edu

Michael S. Bernstein
Stanford University
Stanford, CA, USA
msb@cs.stanford.edu

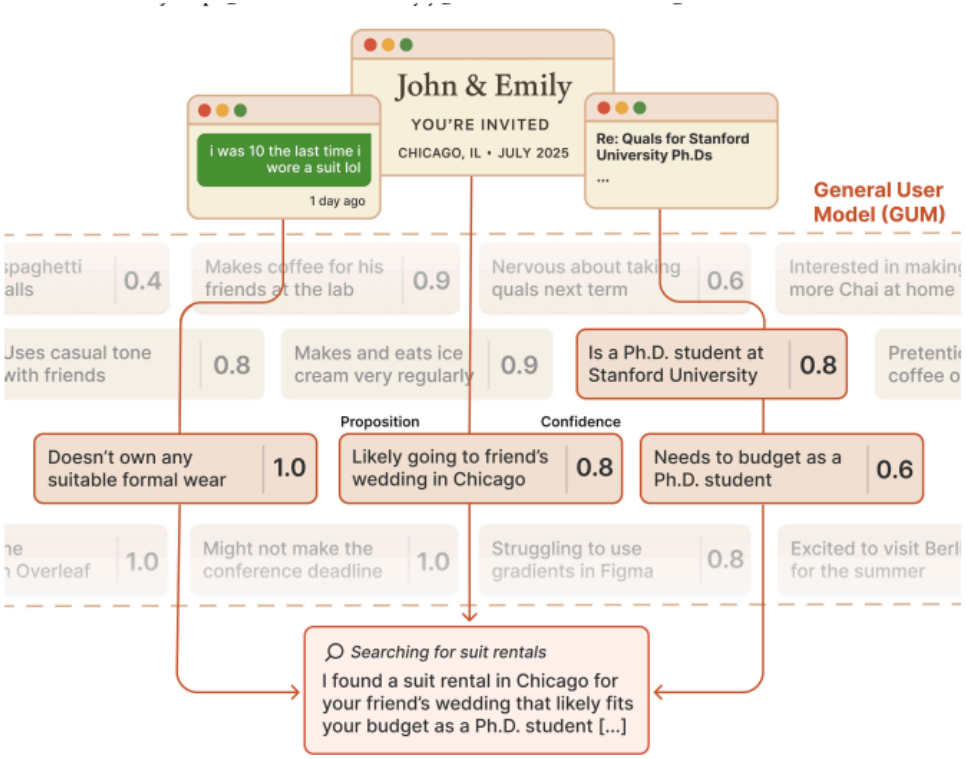


Figure 1: General User Models take as input completely unstructured interaction data (top), transforming these observations into structured propositions about a user (center). Together, these propositions create a *general user model* (GUM). We instantiate our user model in an assistant (GUMBO) that proactively discovers and executes suggestions on a user's behalf (bottom).

User Modeling

Supporting Reuse by Delivering Task-Relevant and Personalized Information

Yunwen Ye[1,2]
[1]SRA Key Technology Laboratory, Inc.
3-12 Yotsuya, Shinjuku, Tokyo 160-004, Japan
+1-303-492-8136
yunwen@cs.colorado.edu

Gerhard Fischer[2]
[2]Department of Computer Science
University of Colorado
Boulder, CO80303-0430, USA
+1-303-492-1592
gerhard@cs.colorado.

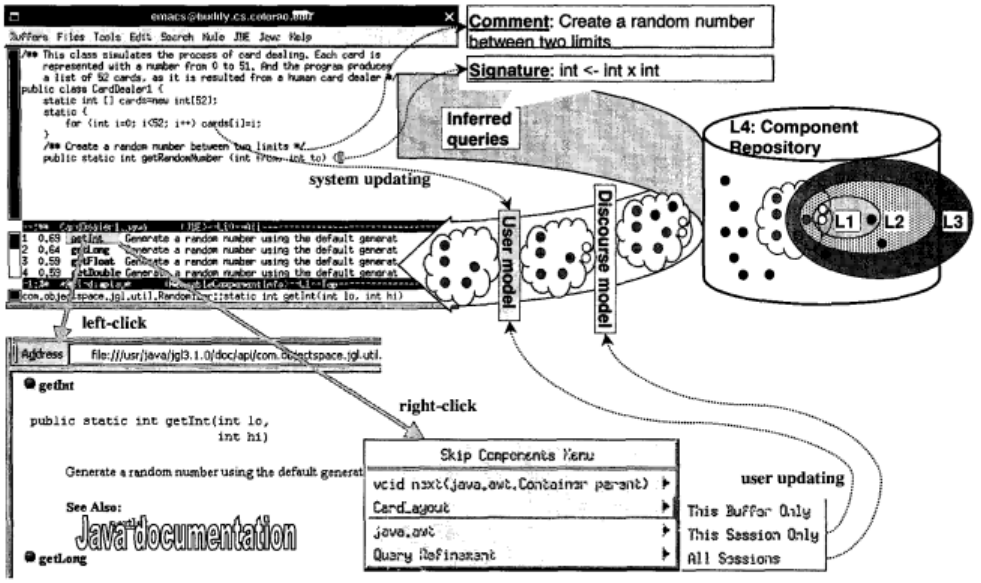


Figure 3: The system architecture of *CodeBroker*

Components that match the queries, which are extracted from doc comments and signatures, are delivered after being filtered with discourse models and user models. Discourse models (see Section 3.2.3) remove irrelevant components (black dots), and user models (see Section 3.3) remove known components (unshaded dots). Discourse models and user models can both be updated by users through the `Skip Components Menu`. User models are also automatically updated when the system detects the reuse of a component in the workspace. Users who want to know more about a component can go to the Java documentation by clicking on the delivered component.

SQLucid: Grounding Natural Language Database Queries with Interactive Explanations

Yuan Tian
Purdue University
West Lafayette, IN, USA
tian211@purdue.edu

Jonathan K. Kummerfeld
University of Sydney
Sydney, Australia
jonathan.kummerfeld@sydney.edu.au

Toby Jia-Jun Li
University of Notre Dame
Notre Dame, IN, USA
toby.j.li@nd.edu

Tianyi Zhang
Purdue University
West Lafayette, IN, USA
tianyi@purdue.edu

End Users

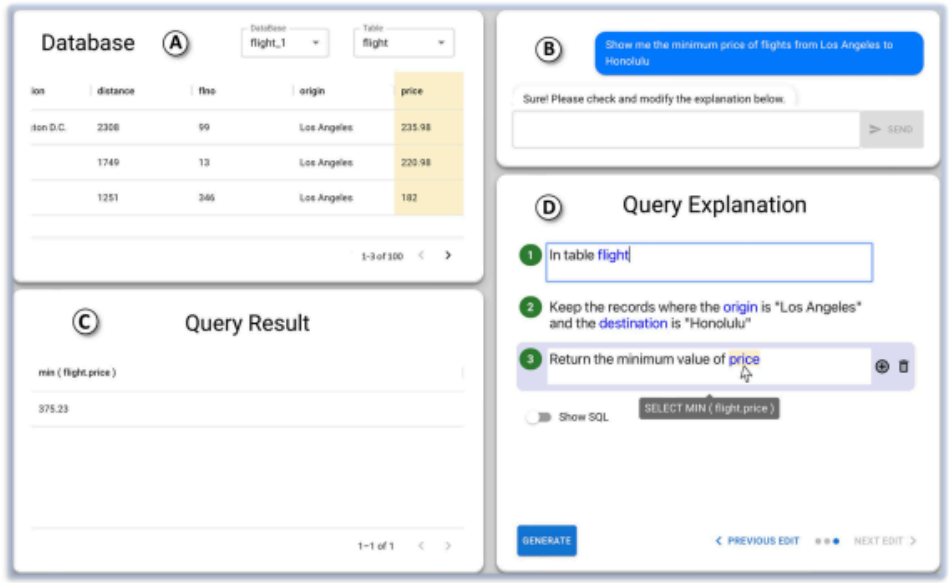


Figure 3: The user interface of SQLucid. (A) The *Database* panel allows users to switch databases and tables in a database. It also allows users to manually inspect, search, and filter data. (B) The *Question* panel allows users to ask a question to the database in natural language. (C) The *Query Result* panel shows the query result as well as the intermediate result of individual steps when the user clicks each step. (D) The *Query Explanation* panel renders the step-by-step SQL explanation in natural language. Users can directly edit the explanation to fix the incorrect behavior in a step, add new steps, or delete existing steps.

Situated Live Programming for Human-Robot Collaboration

Emmanuel Senft
esenft@wisc.edu
University of Wisconsin–Madison
Madison, Wisconsin, USA

Michael Hagenow
University of Wisconsin–Madison
Madison, Wisconsin, USA

Robert Radwin
University of Wisconsin–Madison
Madison, Wisconsin, USA

Michael Zinn
University of Wisconsin–Madison
Madison, Wisconsin, USA

Michael Gleicher
University of Wisconsin–Madison
Madison, Wisconsin, USA

Bilge Mutlu
University of Wisconsin–Madison
Madison, Wisconsin, USA

End Users

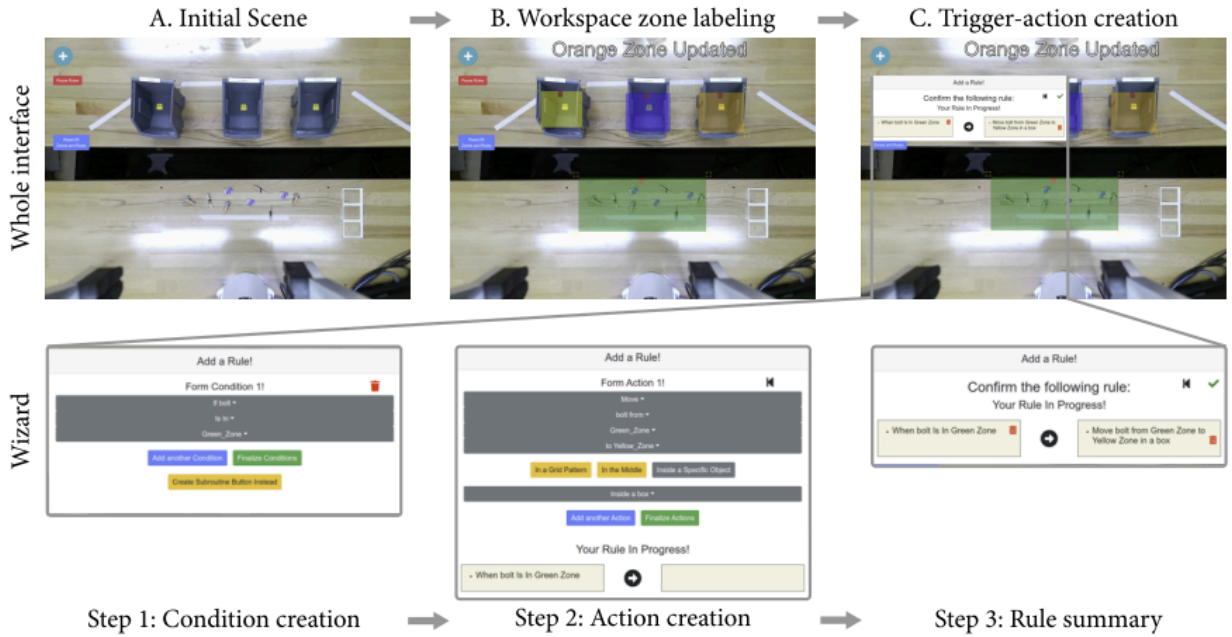


Figure 3: Interface workflow while creating a trigger-action pair to move all of the bolts from the green zone to the storage box in the yellow zone. The top row represents the higher level flow: A. objects are on the robot's table and the boxes on the human's table, with icons showing that the robot detected them. B. The user starts by labeling the interesting areas: one large zone for all the objects and one smaller zone per box. C. The user creates trigger-action pairs. The bottom row presents a focus on the wizard tool to create a trigger-action pair: Step 1: creating the condition: "if a bolt is in the green zone." Step 2: Selecting the action: "move bolt from the green to the yellow zone, in a box." Step 3: Verifying and confirming the pair before adding it to the robot program.

ProgramAlly: Creating Custom Visual Access Programs via Multi-Modal End-User Programming

Jaylin Herskovitz
University of Michigan
Ann Arbor, MI, USA
jayhersk@umich.edu

Andi Xu
University of Michigan
Ann Arbor, MI, USA
andixu@umich.edu

Rahaf Alharbi
University of Michigan
Ann Arbor, MI, USA
rmalharb@umich.edu

Anhong Guo
University of Michigan
Ann Arbor, MI, USA
anhong@umich.edu

End Users

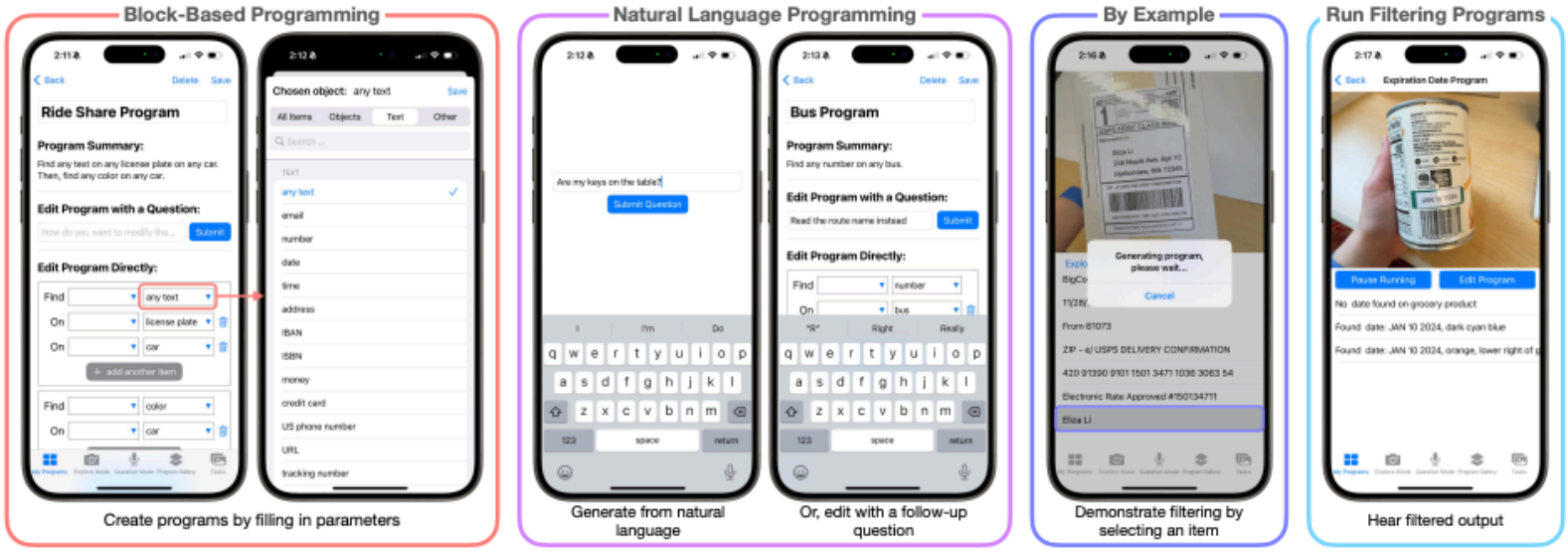


Figure 1: ProgramAlly is an end-user programming tool for creating visual information filtering programs. ProgramAlly provides a multi-modal interface, with block-based, natural language, and programming by example approaches.

Small-Step Live Programming by Example

Kasra Ferdowsifard
UC San Diego
kferdows@eng.ucsd.edu

Allen Ordookhanians
UC San Diego
aordookh@ucsd.edu

Hila Peleg
UC San Diego
hpeleg@eng.ucsd.edu

Sorin Lerner
UC San Diego
lerner@cs.ucsd.edu

Nadia Polikarpova
UC San Diego
npolikarpova@eng.ucsd.edu

End Users

Figure 1. Writing code using SNIPPY: (a)-(d) generates the first statement and (e)-(h) generates the second statement.

Auto-Icon: An Automated Code Generation Tool for Icon Designs Assisting in UI Development

Sidong Feng
Suyu Ma
Faculty of Information Technology
Monash University
Melbourne, Australia
sidong.feng@monash.edu,suyu.ma1@monash.edu

Jinzhong Yu
Alibaba Group
Hangzhou, China
jinzhong.yjz@alibaba-inc.com

Chunyang Chen
Faculty of Information Technology
Monash University
Melbourne, Australia
chunyang.chen@monash.edu

Tingting Zhou
Yankun Zhen
Alibaba Group
Hangzhou, China
miaojing@taobao.com,shadow2597758@gmail.com

UI

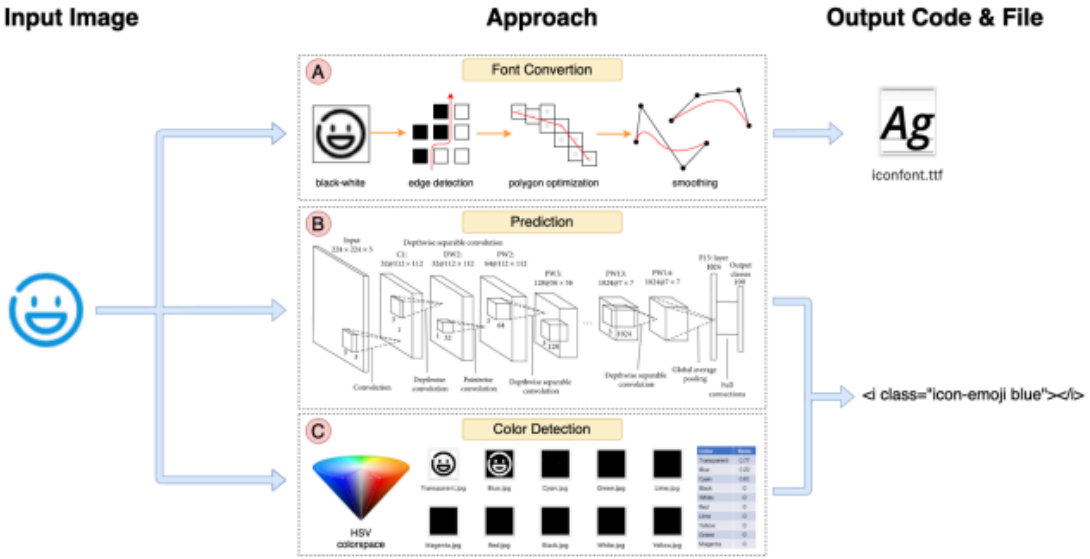


Figure 1: The approach of our tool, *Auto-Icon*, involving font convertion, prediction and color detection.

Sikuli: Using GUI Screenshots for Search and Automation

UI

Tom Yeh *Tsung-Hsiang Chang* *Robert C. Miller*
EECS MIT & CSAIL
Cambridge, MA, USA 02139
{tomyeh,vgod,rcm}@csail.mit.edu

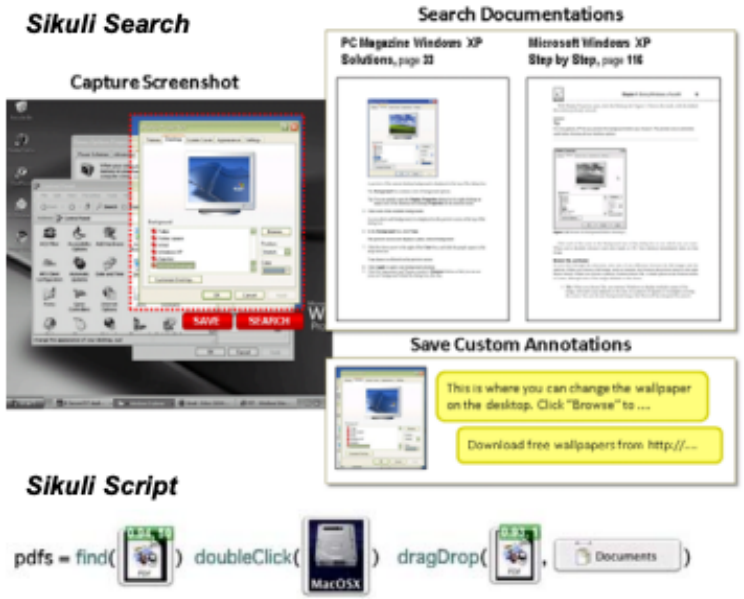
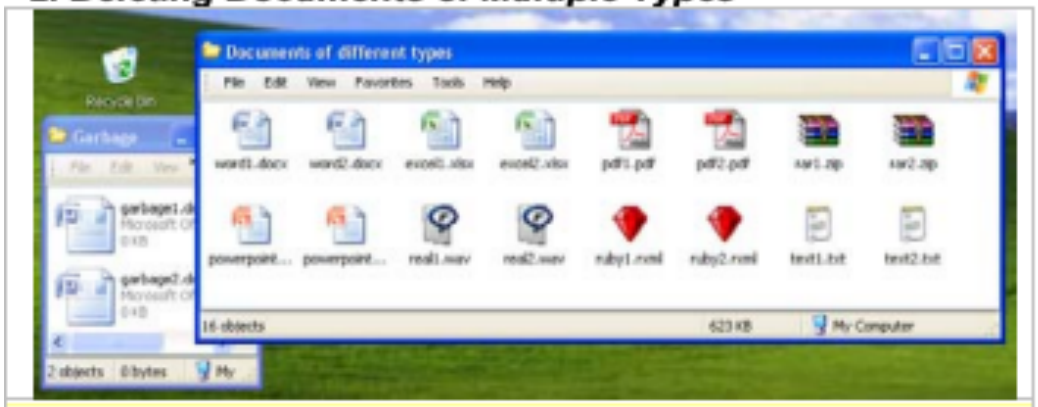


Figure 1: *Sikuli Search* allows users to search documentation and save custom annotations for a GUI element using its screenshot (captured by stretching a rectangle around it). *Sikuli Script* allows users to automate GUI interactions also using screenshots.

2. Deleting Documents of Multiple Types




```
1: def recycleAll(x):
2:     for region in find(x).anySize().regions:
3:         dragDrop(region, )

4: patterns = [ , , ]
5. for x in patterns:
6:     recycleAll(x)
```

Collaboration

Code Space: Touch + Air Gesture Hybrid Interactions for Supporting Developer Meetings

Andrew Bragdon[1,2], *Rob DeLine*[2], *Ken Hinckley*[2], *Meredith Ringel Morris*[2]
[1]Microsoft, [2]Microsoft Research
Redmond, WA, 98052 USA
{anbrag, rdeline, kenh, merrie}@microsoft.com

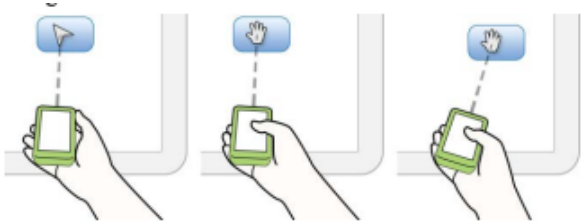


Fig. 4. Pointing and manipulating with a touch-enabled phone.

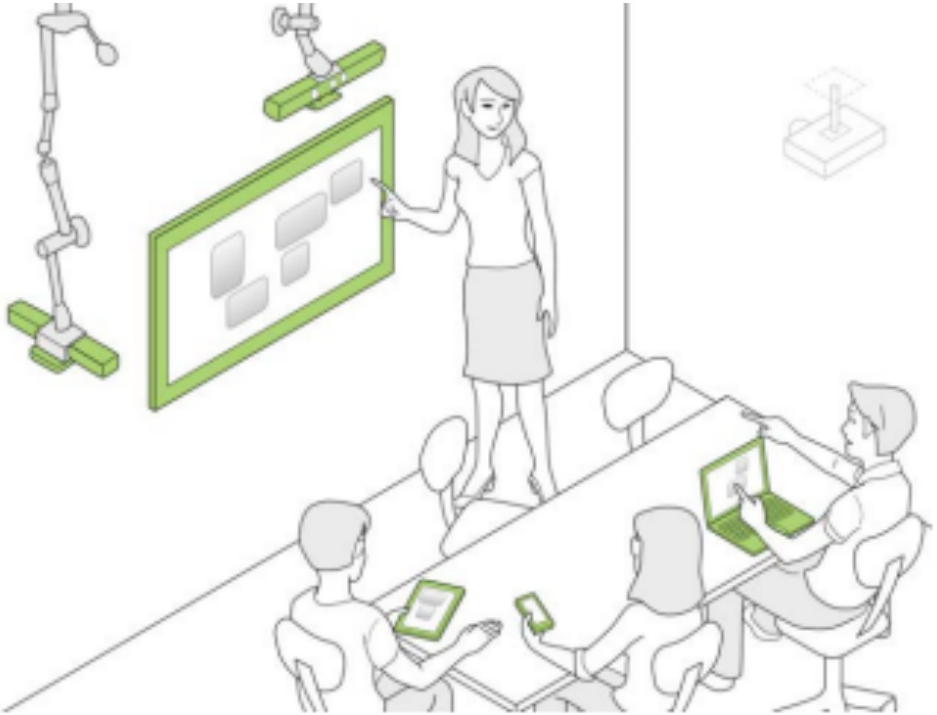


Fig. 1. Code Space meetings include shared multi-touch displays, depth cameras, mobile devices and cross-device interaction

Collaboration

What Would Other Programmers Do? Suggesting Solutions to Error Messages

Björn Hartmann[1], Daniel MacDougall[2], Joel Brandt[2], Scott R. Klemmer[2]

1–Computer Science Division
University of California, Berkeley, CA 94720
bjoern@cs.berkeley.edu

2–Stanford University HCI Group
Computer Science Dept, Stanford, CA 94305
{dmac,jbrandt,srk}@cs.stanford.edu

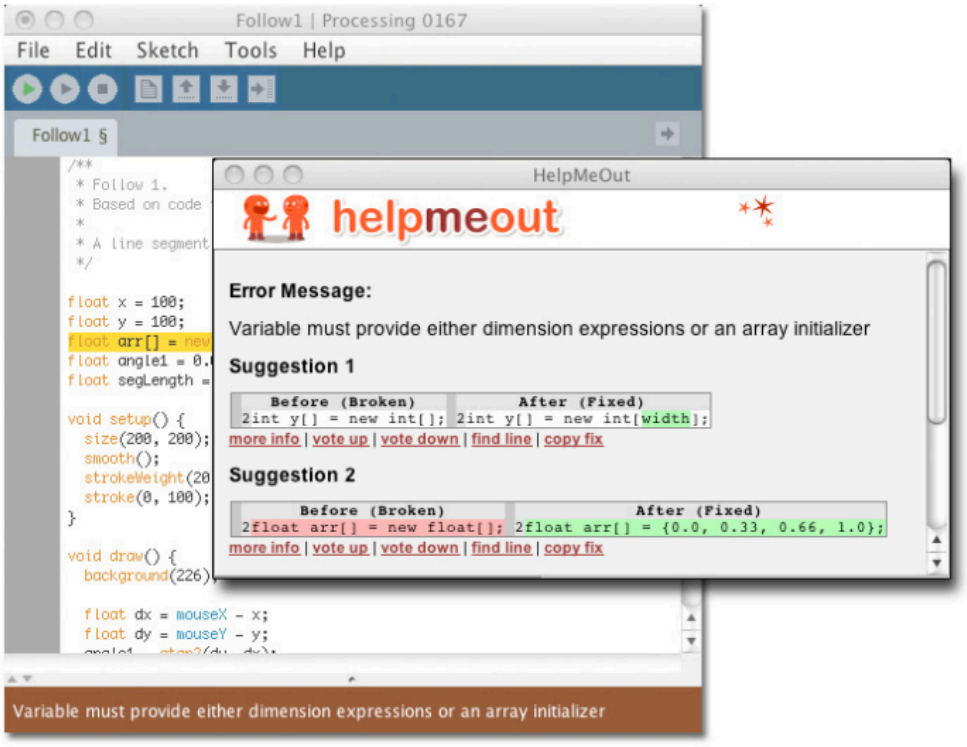


Figure 2. The HelpMeOut Suggestion Panel shows possible corrections for a reported compiler error.

Distinguished Paper

Debugging Reinvented: Asking and Answering Why and Why Not Questions about Program Behavior

Amy J. Ko and Brad A. Myers
Human-Computer Interaction Institute
School of Computer Science, Carnegie Mellon Universi
5000 Forbes Avenue, Pittsburgh, PA 15213
{ajko, bam}@cs.cmu.edu

Debugging

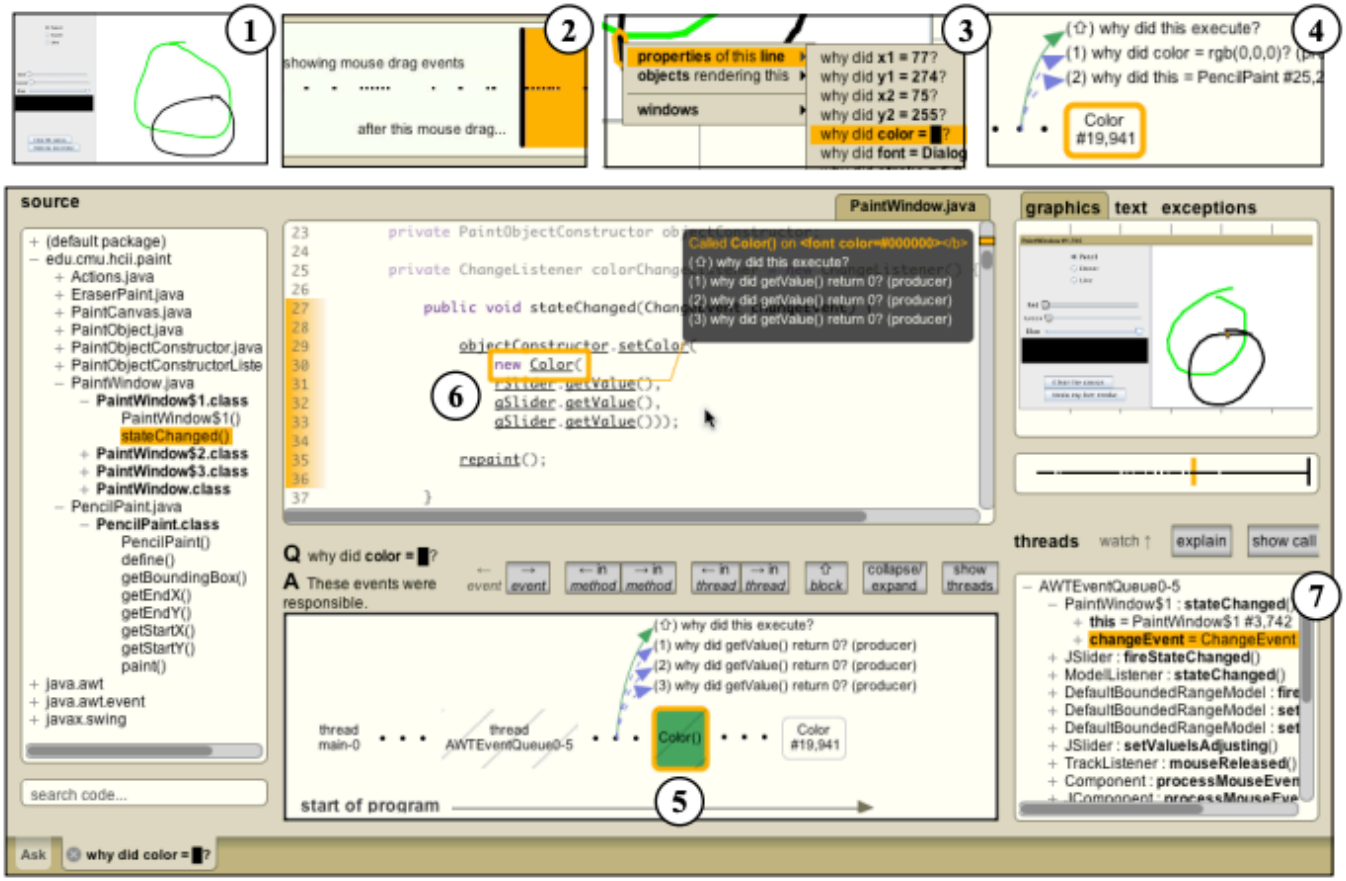


Figure 1. Using the Whyline: (1) The developer demonstrates the behavior; (2) after the trace loads, the developer finds the output of interest by scrubbing the I/O history; (3) the developer clicks on the output and chooses a question; (4) the Whyline provides an answer, which the developer navigates (5) in order to understand the cause of the behavior (6).

NaNofuzz: A Usable Tool for Automatic Test Generation

Matthew C. Davis
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
mcd2@cs.cmu.edu

Sangheon Choi
Rose-Hulman Institute of Technology
Terre Haute, Indiana, USA
chois3@rose-hulman.edu

Sam Estep
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
estep@cmu.edu

Brad A. Myers
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
bam@cs.cmu.edu

Joshua Sunshine
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
sunshine@cs.cmu.edu

Testing

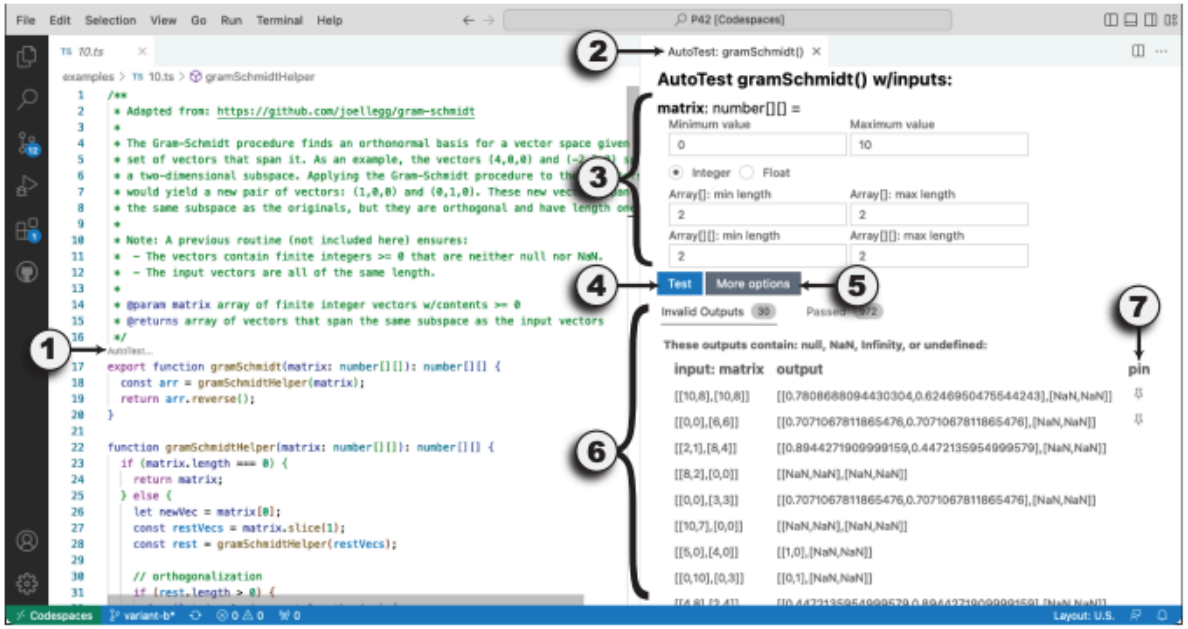


Figure 1: The NaNofuzz user interface in the Visual Studio Code IDE provides one-click test generation for TypeScript programs. Key UI elements: (1) AutoTest button above a function signature, (2) NaNofuzz testing window beside the program under test, (3) Customizable input parameters with default values, (4) Test button to start NaNofuzz, (5) Advanced options, (6) Categorized testing results with likely bugs prioritized in the display, (7) Pin button to add test cases to the test suite in Jest format.

Evaluation Methods

Code bubbles: rethinking the user interface paradigm of integrated development environments

Magentic-UI: Towards Human-in-the-loop Agentic Systems

Quantitative Analysis

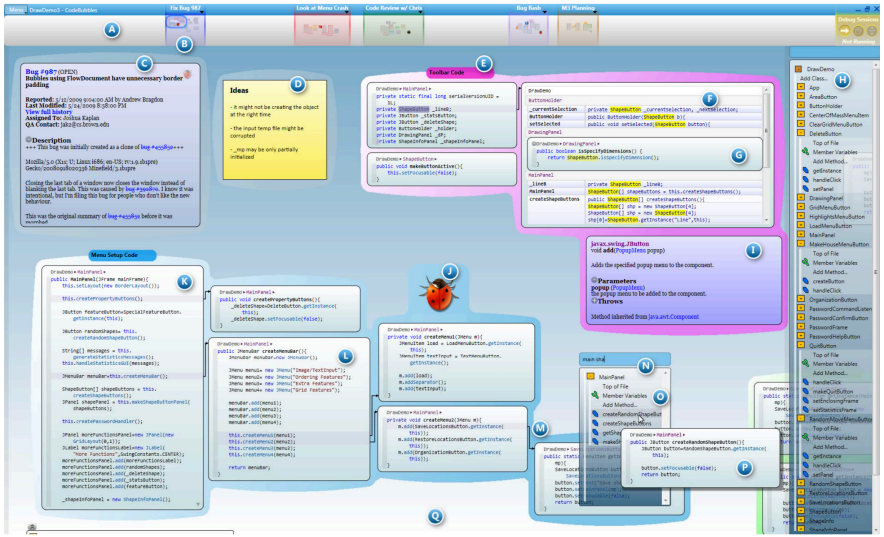


Figure 1. *The Code Bubbles IDE. See section 2 below, Scenario, and section 6 below, IDE User Interface, below. Resolution: 1920x1200 (space reserved for taskbar).*

How many functions can one see on screen simultaneously?
How many UI operations must one use to create concurrently-visible working sets?

Table 1: Performance of Magentic-UI compared to a selection of baselines on the test sets of GAIA and AssistantBench and on WebVoyager and WebGames. The numbers reported denote the exact task completion rate as a percentage. All results for baselines are obtained from either the corresponding benchmark leaderboard, academic papers or blog posts as of July 6 2025.
*: On WebVoyager we ran Magentic-UI with only the WebSurfer agent so that we test only for web browsing abilities.
**: On WebGames, we ran Magentic-UI with only the WebSurfer and FileSurfer using GPT-4o and with two additional tools to the WebSurfer: uploadFile and generalClick (allows for right-click and holding clicks).

Method	GAIA	AssistantBench (accuracy)	WebVoyager	WebGames
Magentic-One (4o, o1)	38.00	27.7	–	–
SPA → CB (Claude) [109]	–	26.4	–	–
Su Zero Ultra (no public info)	80.04	–	–	–
tt_api_1 (GPT-4o) (no public info)	–	28.30	–	–
AWorld (Claude Sonnet-4, Gemini 2.5-pro,4o, DeepSeek-v3) [4]	77.08	–	–	–
Langfun Agent 2.3 [8]	73.09	–	–	–
Claude Computer-Use [3]	–	–	52	35.3
Proxy	–	–	82	43.1
OpenAI Operator [57]	12.3 (4o), 62.2 (o3)	–	87.0	–
GPT-4o (SoMs+ReAct) [83]	6.67 (no tools)	16.5 (no tools)	64.1 [28]	41.2
Browser Use [54]	–	–	89.1	–
Human	92.00	–	–	95.7
Magentic-UI (o4-mini)	42.52	27.6	82.2*	45.5**

NaNofuzz: A Usable Tool for Automatic Test Generation

Generative Agents: Interactive Simulacra of Human Behavior

Qualitative Analysis

We analyzed free-form post-survey data qualitatively using the **inductive thematic analysis procedure**
…
the third author established replicability of the second author's result by re-coding the data for five random participants using the second author's codes. This resulted in a substantial [36] level of **inter-rater reliability** (= 0.6962).

T1: Automation can reduce human cognitive effort required for creating test cases.
T2: Automation can reduce manual labor for creating tests.
T3: Flexibility is valuable—when I need it.
T4: I need to specify what correctness means.
T5: Building a test suite can require iteration and exploration.
T6: Understanding many test outputs helps me understand the program behavior and be more confident.
T7: Intuitive tool design can reduce barriers.

Furthermore, the first author conducted an inductive analysis [95] to study the qualitative distinctions between the responses produced in each condition. We employed qualitative open coding [33] in two phases. In the frst phase, we **generated codes** that closely represented the generated responses at the sentence level. In the second phase, we **synthesized the resulting codes** from the frst phase to extract higher-level themes. We utilized these themes to compare the types of responses generated in our study.

Grounded Theory

Table 1. Participants overview. PCU: Prior Copilot Usage. We show the language(s) used on their task, their usage experience with their task language (Never, Occasional, Regular, Professional), whether they had used Copilot prior to the study, their occupation, and what task they worked on.

ID	Language(s)	Language Experience	PCU	Occupation	Task
P1	Python	Professional	Yes	Professor	Chat Server
P2	Rust	Professional	No	PhD Student	Chat Client
P3	Rust	Occasional	No	Professor	Chat Client
P4	Python	Occasional	Yes	Postdoc	Chat Server
P5	Python	Regular	No	Software Engineer	Chat Client
P6	Rust	Professional	Yes	PhD Student	Chat Server
P7	Rust	Professional	No	Software Engineer	Chat Server
P8	Rust	Professional	No	PhD Student	Chat Server
P9	Rust[1]	Occasional	No	Undergraduate Student	Benford's law
P10	Python	Occasional	No	Undergraduate Student	Chat Client
P11	Rust+Python	Professional + Professional	Yes	Cybersecurity Developer	Benford's law
P12	Rust+Python	Professional + Occasional	Yes	Software Engineer	Benford's law
P13	Rust+Python	Regular + Occasional	Yes	PhD Student	Benford's law
P14	Python	Professional	No	PhD Student	Advent of Code
P15	Python	Professional	Yes	PhD Student	Advent of Code
P16	Haskell	Professional	No	PhD Student	Advent of Code
P17	Rust	Professional	Yes	Founder	Advent of Code
P18	Java	Occasional	No	PhD Student	Advent of Code
P19	Python	Occasional	No	PhD Student	Advent of Code
P20	Haskell	Occasional	Yes	PhD Student	Advent of Code

As opposed to evaluating fixed, a priori hypotheses, a study using **the GT methodology seeks to generate new hypotheses in an overarching theory developed without prior theoretical knowledge on the topic.** A researcher produces this theory by constantly interleaving data collection and data analysis.

…

We began our study with the blank slate question: "How do programmers interact with Copilot?" Our bimodal theory of acceleration and exploration was not yet formed. During each session, we took notes to guide our semi-structured interview.

Why Jonny Can't Prompt: How Non-AI Experts Try (and Fail) to Design LLM Prompts

Design Probe

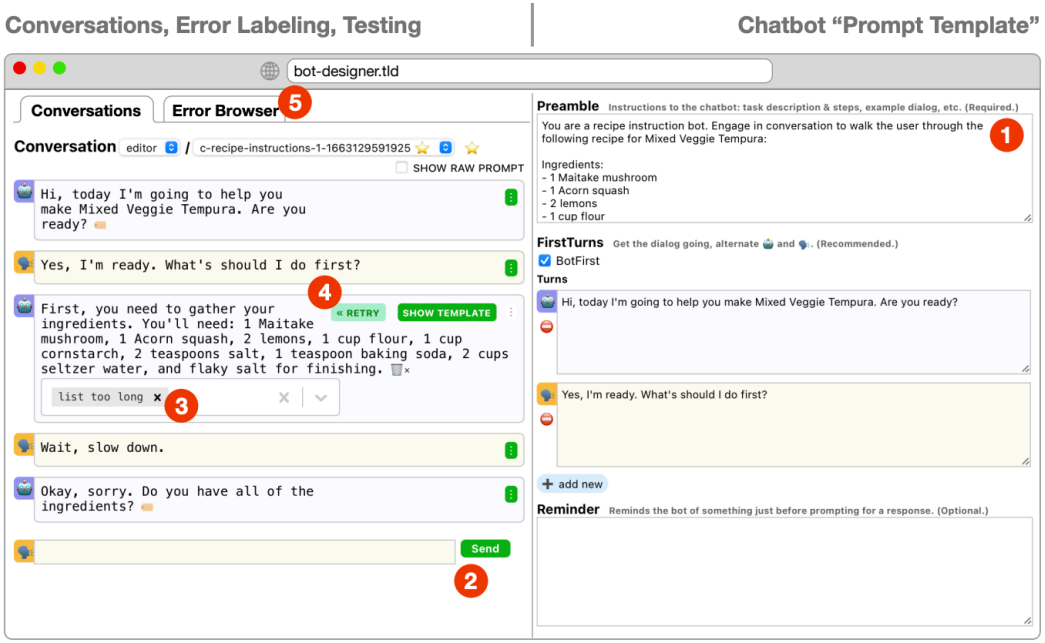


Figure 1: BotDesigner main user interface. Right pane: the *prompt template* authoring area (1); designers use plain text here to describe desired behavior for the chatbot they are developing. Left, tabbed pane: the conversation authoring area; here, designers can send chat messages (2) to the bot defined by the current template on the right. A *label* button reveals a labeling widget (3) that allows designers to attach an arbitrary text label to any bot-produced chat message, for aggregation and later identification. A *retry* button (4), only visible after the *prompt template* has been changed, lets designers test what new bot chat message would be produced, at that specific point in the conversation, as a result of the prompt template change. The "Error Browser" tab (5) reveals a UI panel for designers to test prompt changes against all labeled bot responses, across all conversations (see Figure 2).

We want to dive deep into what those struggles reveal about people's assumptions and understanding of prompt-based systems in order to inform **how end-user-facing prompt design tools might best support their users.**

…

We developed a no-code prompt design tool, BotDesigner, as a design probe [6], and conducted a user study with components of contextual inquiry and interview. We chose to create a design probe because the robustness of prompt strategies is highly dependent on specifc conversational contexts, as is users' ability to create prompts and debug. **Enabling people to engage in designing LLM-and-prompt-based chatbots hands-on offers deeper insights than conducting interviews about hypothetical scenarios alone.**

Context-Aware Conversational Developer Assistants

NaNofuzz: A Usable Tool for Automatic Test Generation

User study

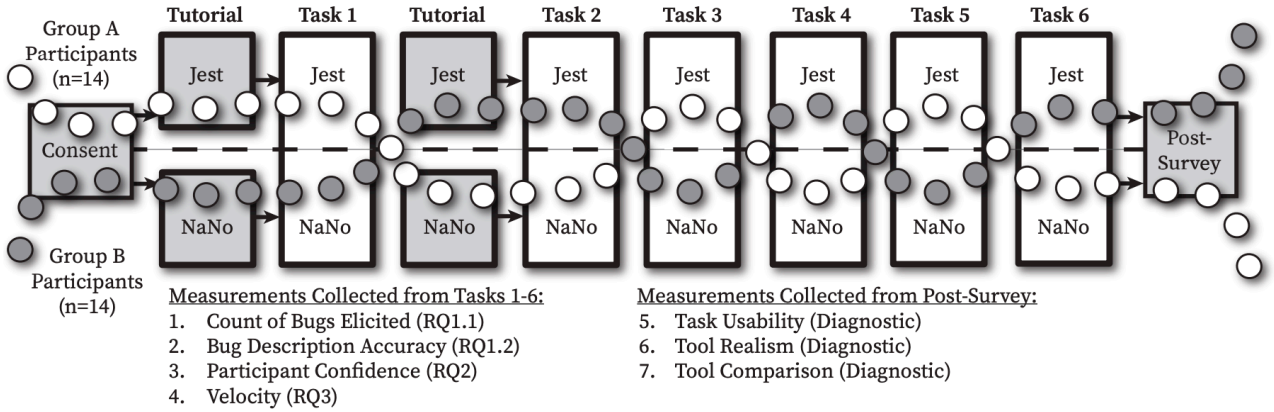


Figure 3: Visualization of study task sequence (see Section 5). Participants are randomly assigned by pairs into Group A or Group B, which determines the treatment for each task. Shaded tasks (e.g., tutorials) are not time-limited.

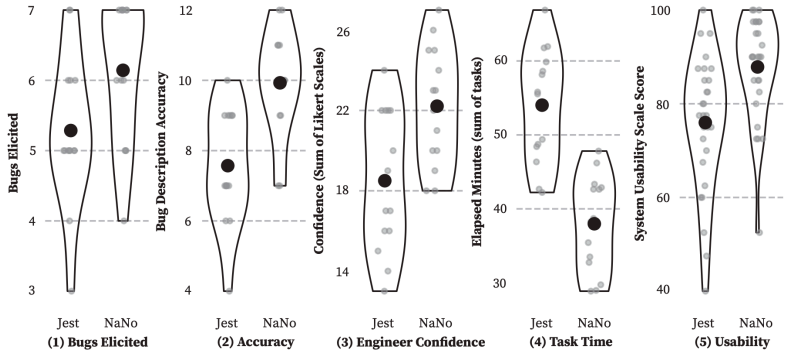


Figure 5: Violin plots showing the distribution of results by treatment for Measures 1-5. Y-axis for Measures 1-4 shows the sum of six tasks for paired participants. Y-axis for Measure 5 is the SUS [9] score range. The mean is indicated by a black dot. Grey dots indicate observations. Wider areas of the plot indicate a larger number of observations.

Table 3: Order, sample questions, and tasks from our mixed methods study.

Interview - Part One	
1.1	Walk me through typical development tasks you work on every day.
1.2	How do you choose a work item; what are the steps to complete it?
1.3	How do you debug a failed test?
2	To help you get familiar with Alexa, ask Alexa to tell us a joke.
3	Can you think of any tasks that you would like to have "magically" completed by either talking to Alexa or by typing into a natural language command prompt?
Experiment - Interaction Task (T1)	
Complete the following tasks:	
	Launch Devy by saying "Alexa, launch Devy" [..]
T1.1	Using Devy, try to get the name of the person whom you might contact to get help with making changes to this 'readme' file.
T1.2	Next, make sure you are on branch 'iss2' and then make a change to this 'readme' file (and save those changes).
T1.3	Finally, make those changes available on GitHub.
Experiment - Demonstration Task (T2)	
Complete the following tasks:	
T2.1	Say "Alexa, tell Devy to list my issues." to list the first open issue on GitHub. List the second issue by saying "Next", then stop by saying "Stop". Notice that the listed issues are for the correct repository.
T2.2	Say "Alexa, tell Devy I want to work on issue 2." to have Devy prepare your workspace for you by checking out a new branch.
T2.3	Resolve the issue: comment out the debug console.log on line 8 of `log.ts` by prepending it with //. Save the file.
T2.4	Say "Alexa, tell Devy I'm done." to commit your work and open a pull request. Devy will ask if you want to add the new file; say "Yes". Next, Devy recommends up to 3 reviewers. You choose any you like. When completed, Devy will say it created the pull request and open a browser tab showing the pull request. Notice the reviewers you specified have been added. Also, notice that tests covering the changes were automatically run and the test results were included in a comment made by Devy.
Interview - Part Two	
1	Imagine that Devy could help you with anything you would want, what do you think it could help you with and where would it provide most benefit?
2	Are there any other tasks / goals / workflows that you think Devy could help with, maybe not just restricted to your development tasks, but other tools you or your team or your colleagues use?
3	When you think about the interaction you just had with Devy, what did you like and what do you think could be improved.
4	Did Devy do what you expected during your interaction? What would you change?
5	Do you think that Devy adds value? Why or why not?

Administrivia

Final Presentation (Dec 1 & Dec 3)

- 10 minutes per presentation with 3 minutes for questions and discussion.
- Presentation order will be random
- In-class activity: Q&A after each presentation

Final Report (+Implementation) (Due Dec 10)

- 10% for holistic evaluation
- Please refer to the papers we read in designing and writing evaluation plans and the threats to validity section.
- No results section needed.