Live Programming

CS 695 / SWE 699: Programming Tools

Fall 2023

Today

- Part 1 (Lecture)(~45 mins)
  - 10 min break!
- Part 2: One or Two Tech Talks (30 mins)
  - CodePen
  - Google Colab (?)
- Part 3: (In-Class Activity)(60 mins)

Logistics

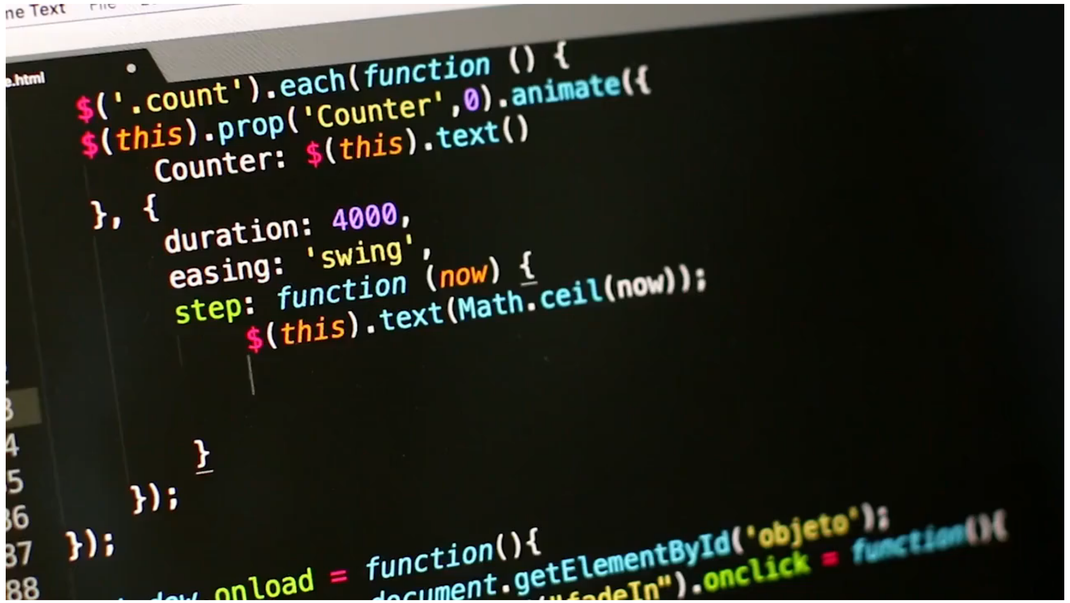
- HW 4 checkpoint due 11/1
- HW 4 due 11/29
- Tech talks should now all be on the dates originally scheduled

Overview

- What is live programming?
- Tools to make programming more live
- Computational Notebooks

Developers work in cycles

Edit



Run

Tools support

Live programming environment

Live programming environments
designed to enable fluid experience

Live programming environments
designed to enable fluid experience

Frequent and short.

Live programming environments
designed to enable fluid experience

Frequent and short.

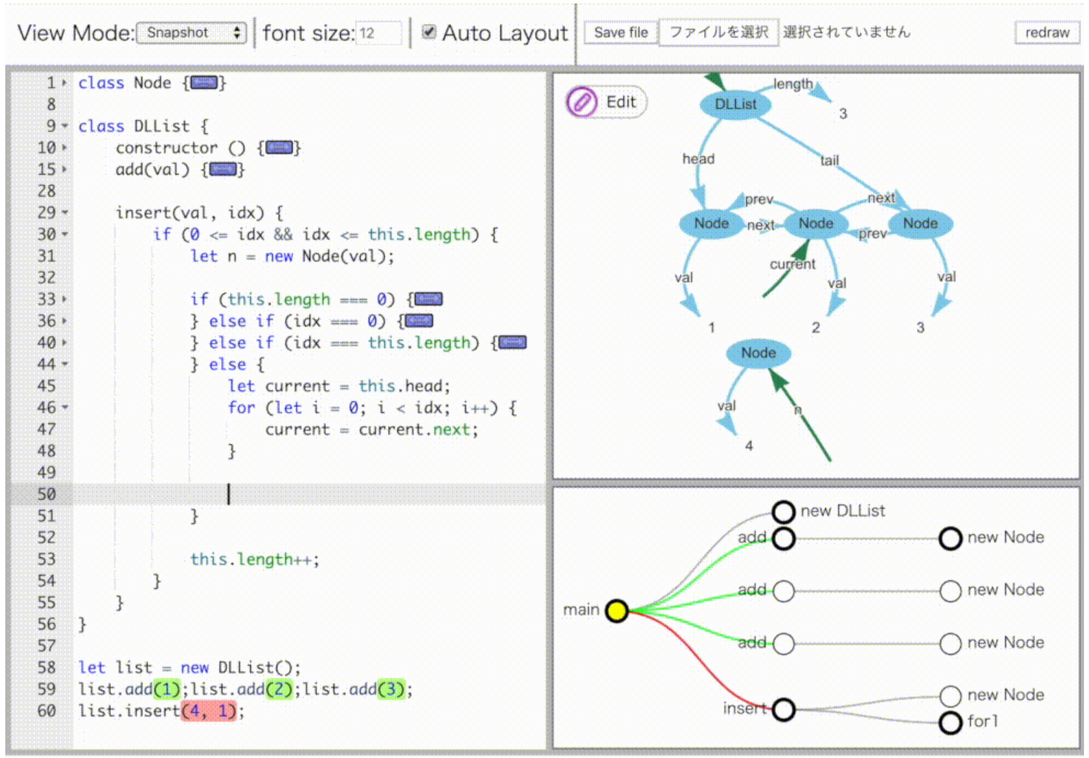
Focus on the edit step.

Live programming environments
designed to enable fluid experience

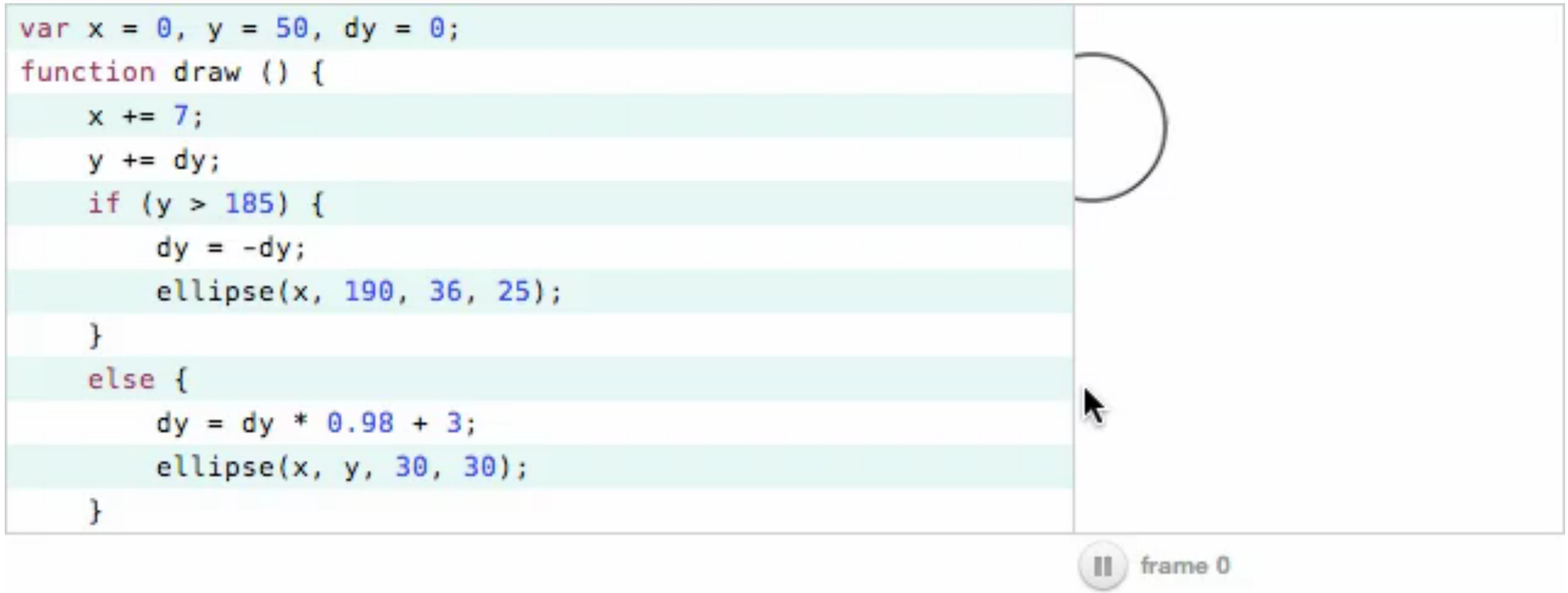
Frequent and short.

Focus on the edit step.

No interruptions.

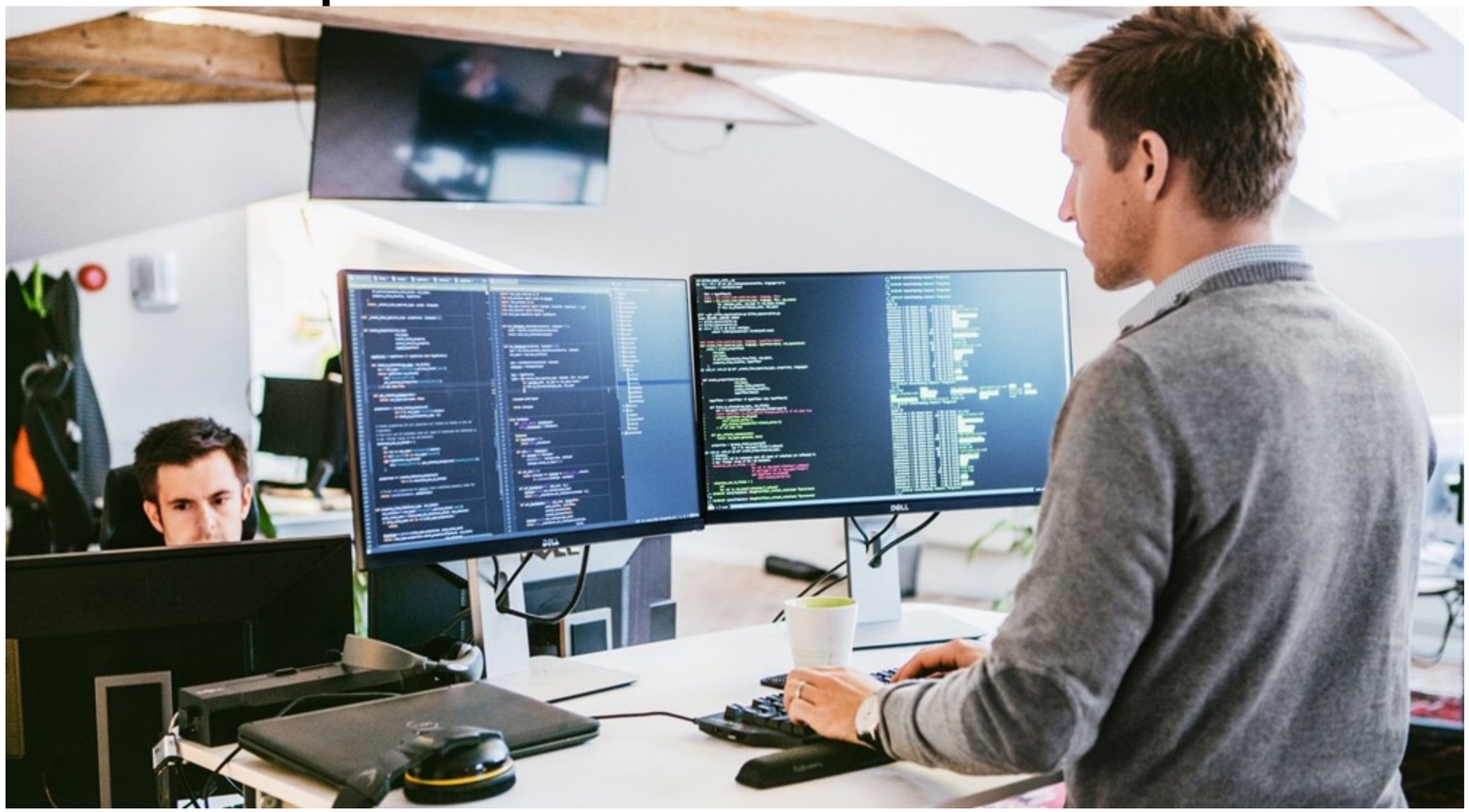


Masuhara et al, Programming Experiences with a Live Programming Environment for Data Structures



Bret Victor, LEARNABLE PROGRAMMING

What is developers’ current edit-run behavior?



Fluidity in current edit-run cycles

Frequent and short.

Focus on the edit step.

No interruptions.

Fluidity in current edit-run cycles

Frequent and short.

RQ1: How long and frequent are edit-run cycles?

Focus on the edit step.

No interruptions.

Fluidity in current edit-run cycles

Frequent and short.

RQ1: How long and frequent are edit-run cycles?

Focus on the edit step.

RQ2: How do developers edit and run?

No interruptions.

Fluidity in current edit-run cycles

Frequent and short.

RQ1: How long and frequent are edit-run cycles?

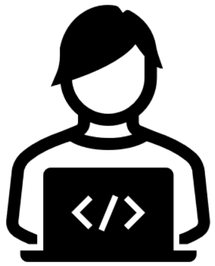
Focus on the edit step.

RQ2: How do developers edit and run?

No interruptions.

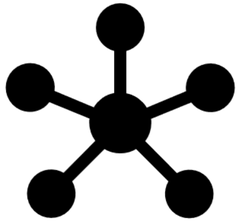
RQ3: How sequential are edit-run cycles, and what causes gaps within and between cycles?

Observe-dev dataset



11 Professional Developers

15 hours of debugging
13 hours of programming



2135 activities in debugging
1368 activities in programming

Observe-dev dataset

Activities

Browsing a file of code Editing a file of code Testing Program Inspecting Program Consulting Resources

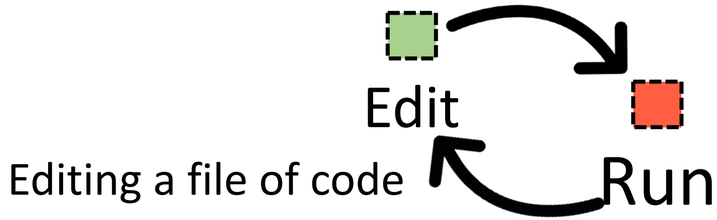
Others

Observe-dev dataset

Activities

Consulting Resources

Others



Editing a file of code

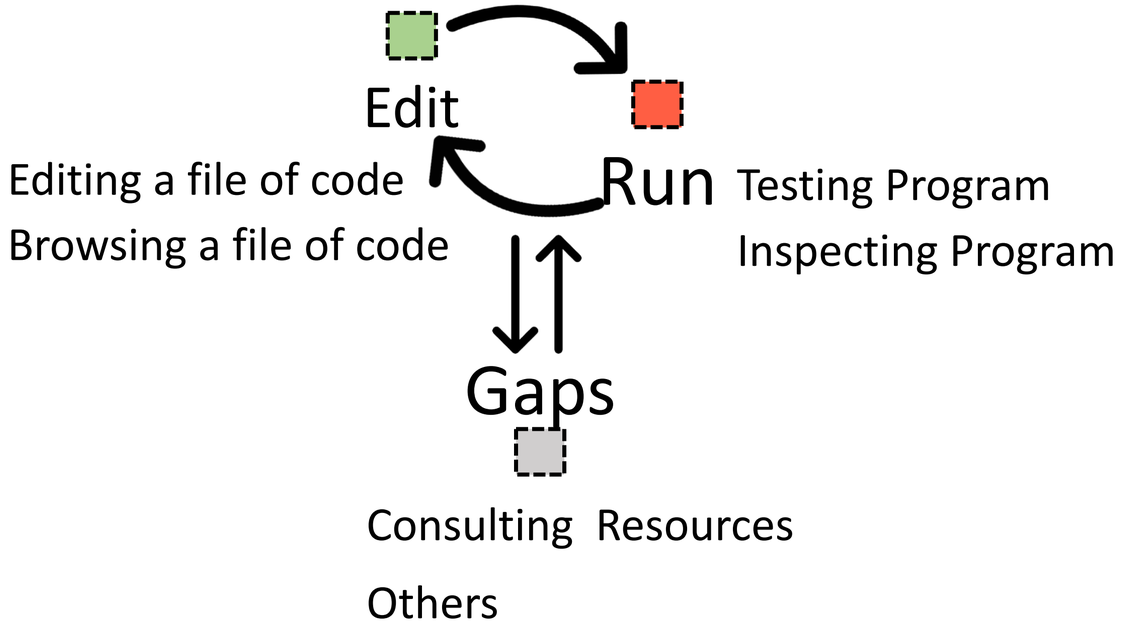
Browsing a file of code

Testing Program

Inspecting Program

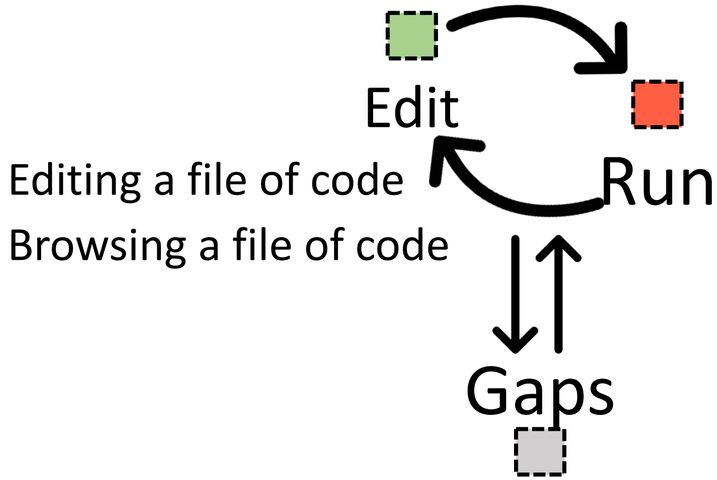
Observe-dev dataset

Activities



Observe-dev dataset

Activities



Editing a file of code

Browsing a file of code

Testing Program

Inspecting Program

Consulting Resources

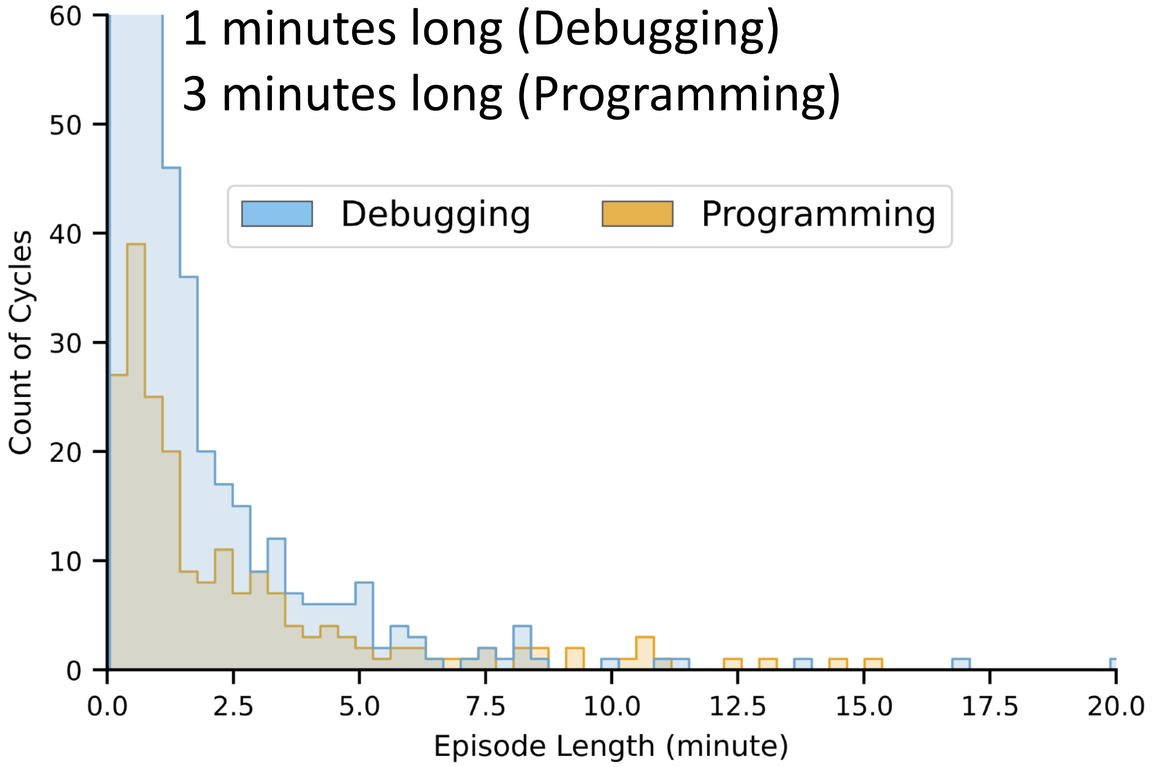
Others

581 cycles in debugging.

207 cycles in programming.

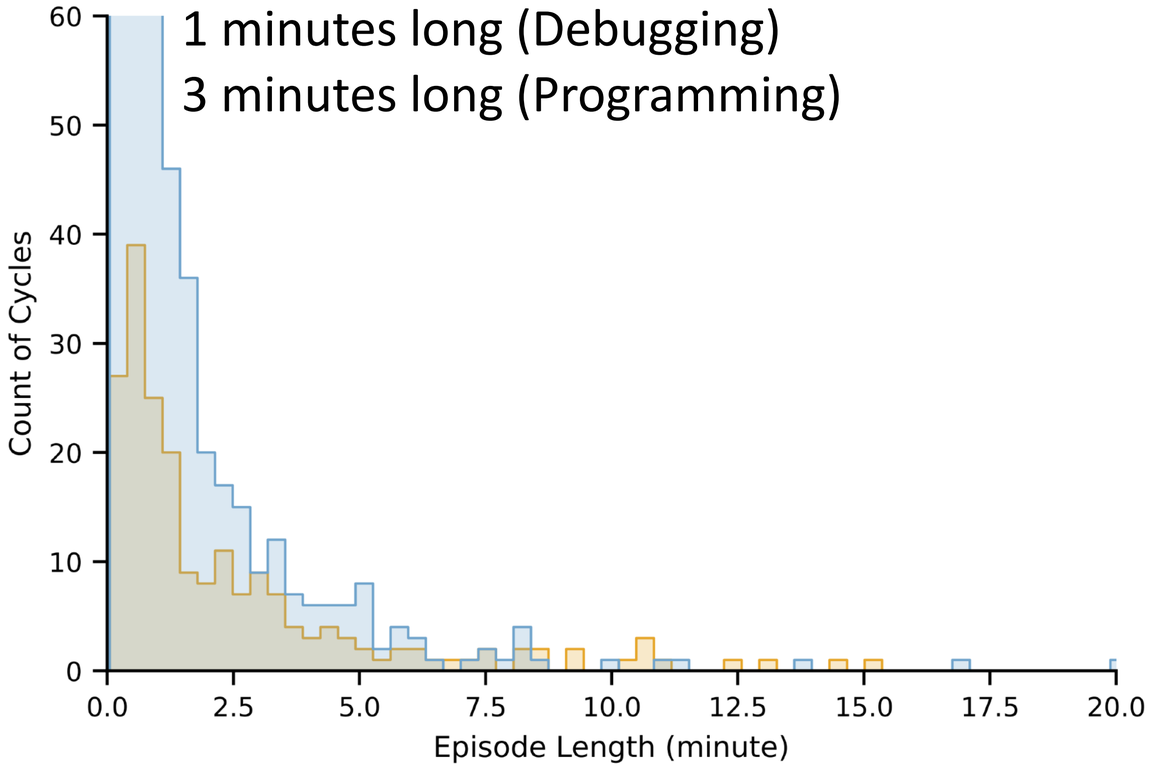
Fluidity in current edit-run cycles

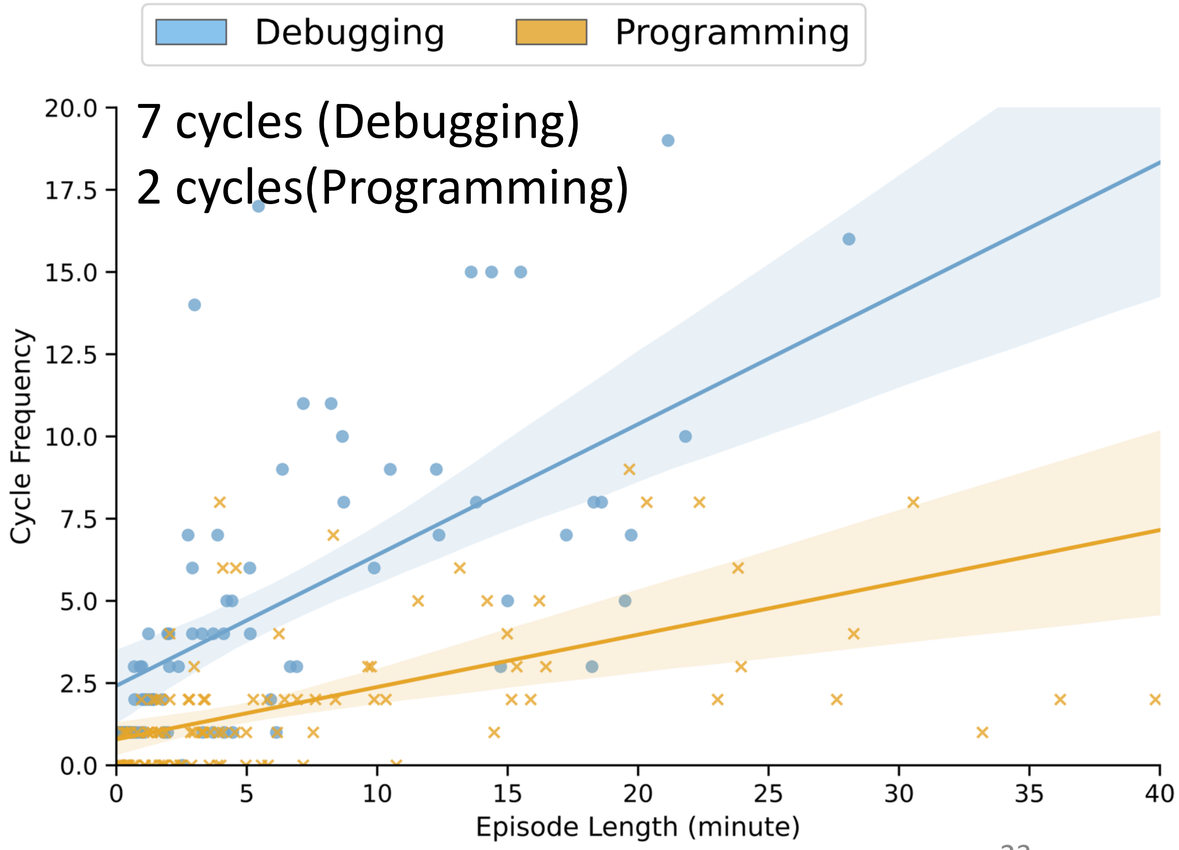
RQ1: How long and frequent are edit-run cycles?



Fluidity in current edit-run cycles

RQ1: How long and frequent are edit-run cycles?



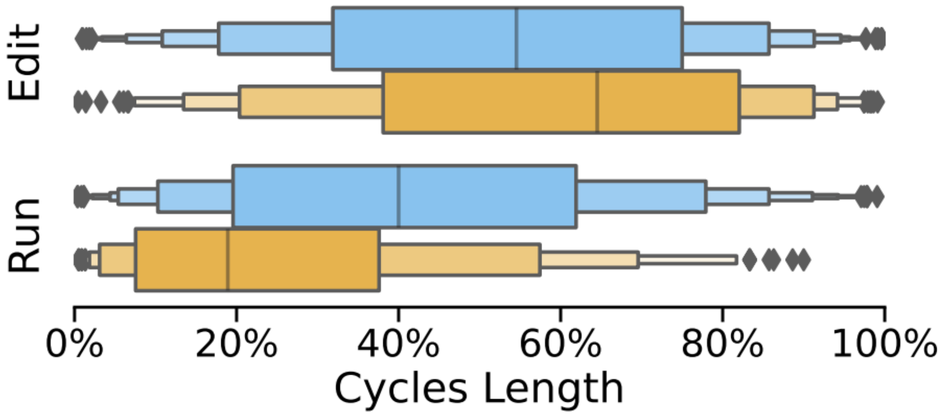


Fluidity in current edit-run cycles

RQ2: How do developers edit and run?

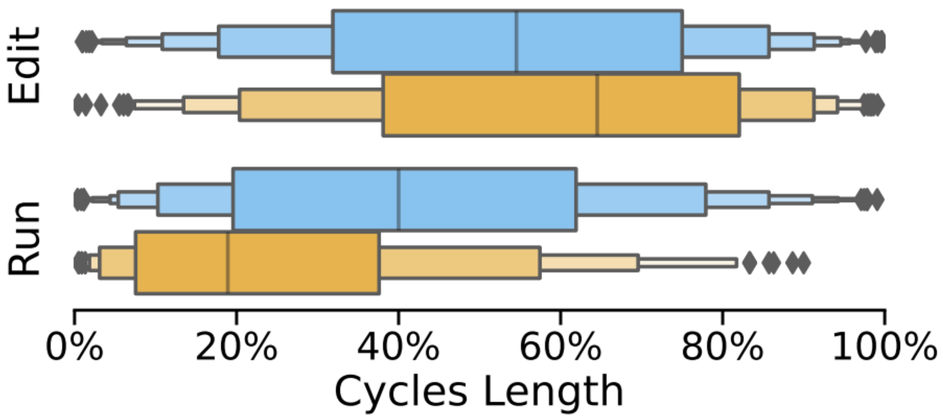
Fluidity in current edit-run cycles

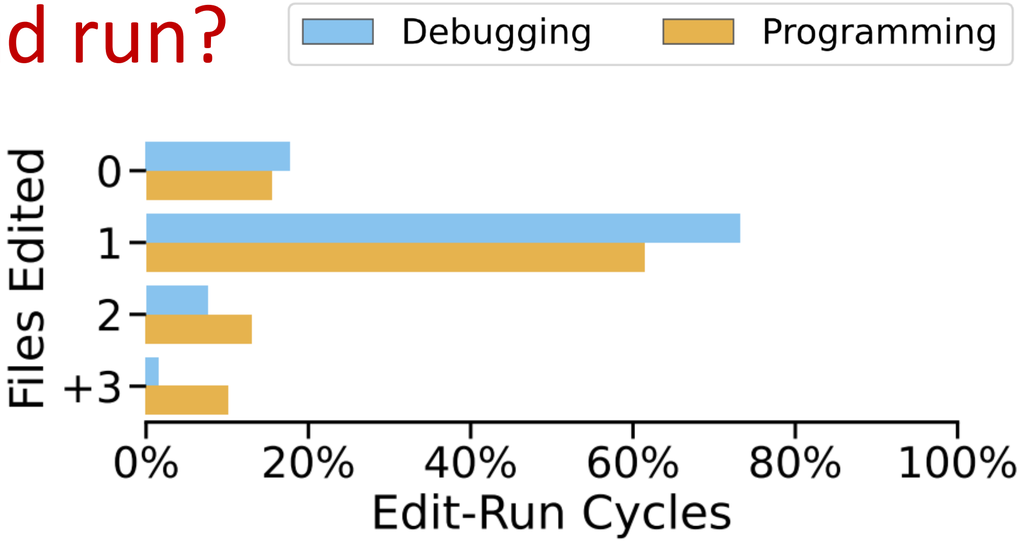
RQ2: How do developers edit and run?



Fluidity in current edit-run cycles

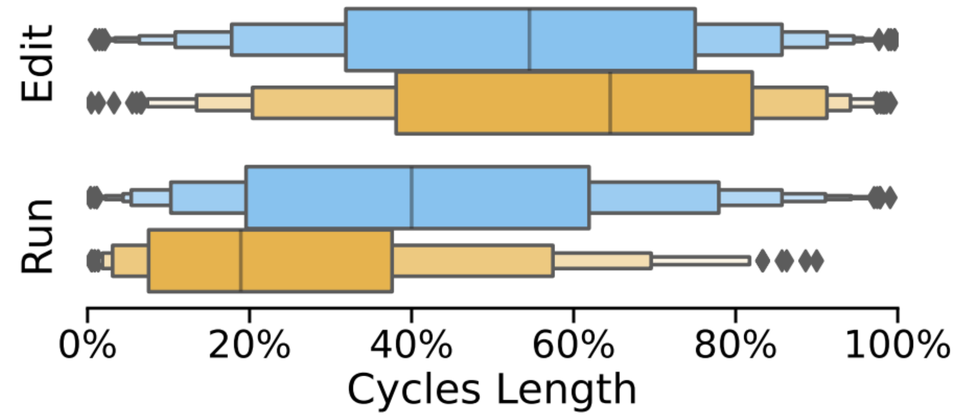
RQ2: How do developers edit and run?

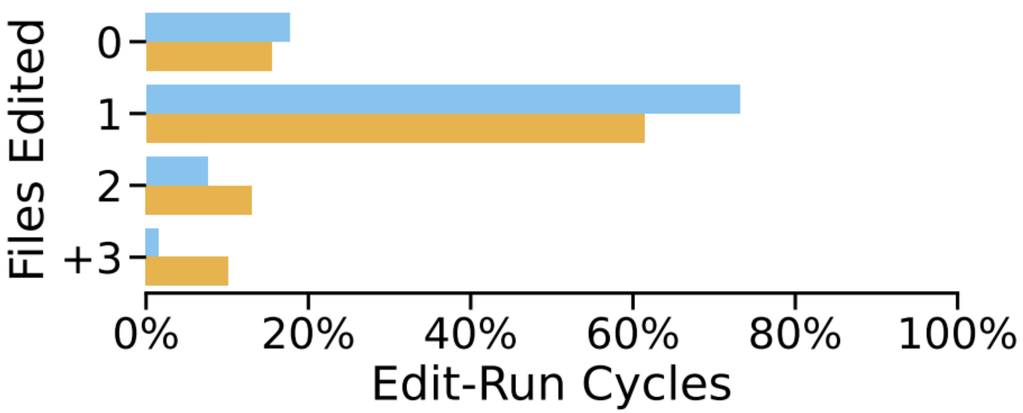


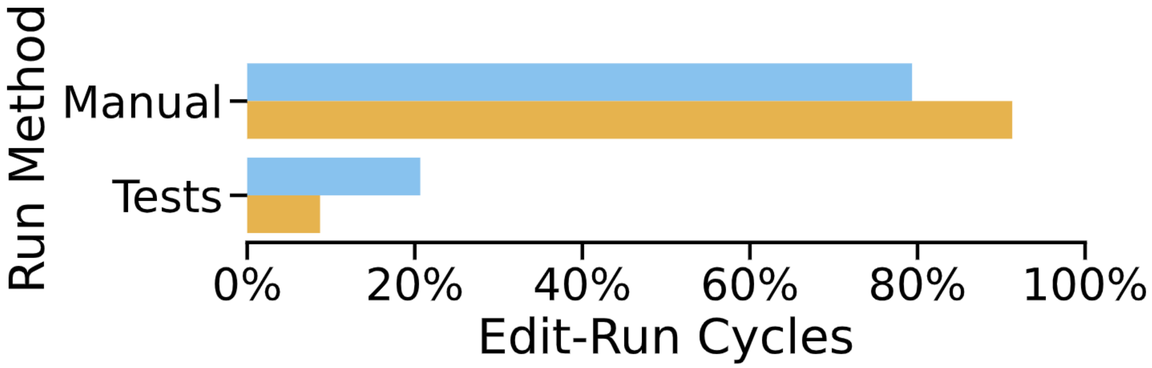


Fluidity in current edit-run cycles

RQ2: How do developers edit and run?

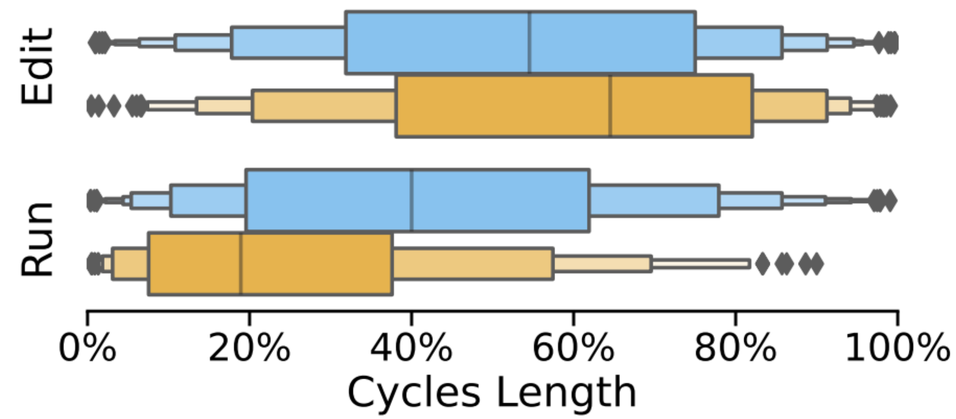


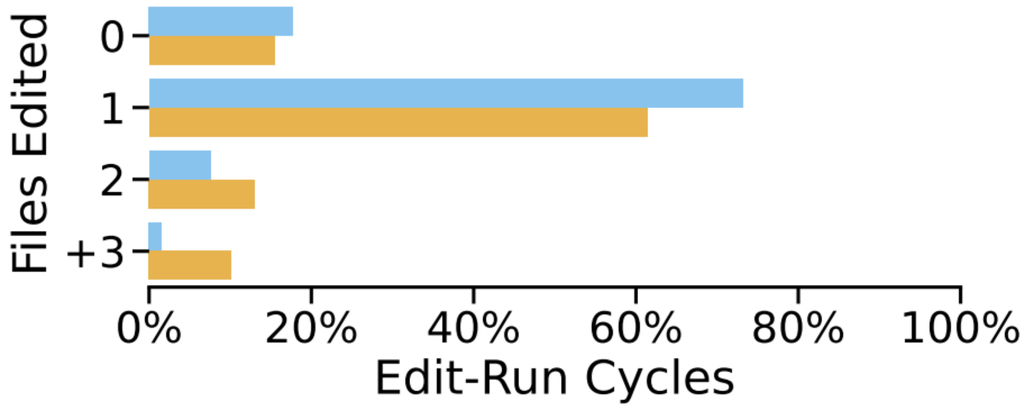


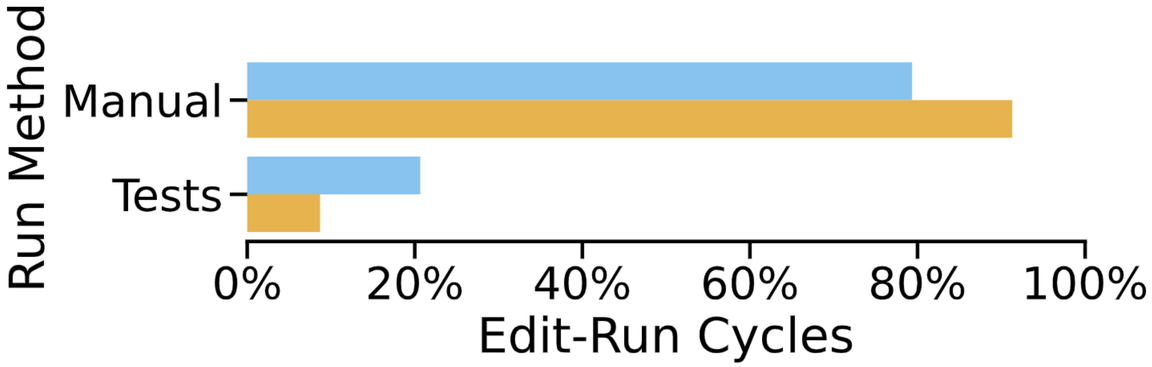


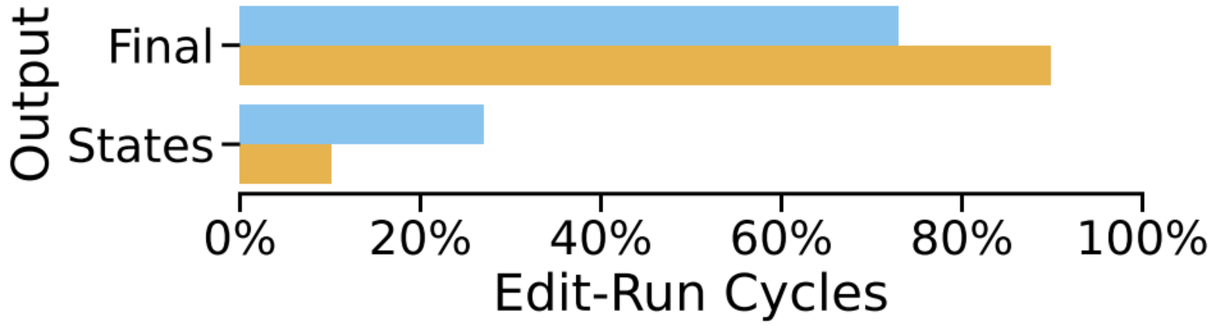
Fluidity in current edit-run cycles

RQ2: How do developers edit and run?







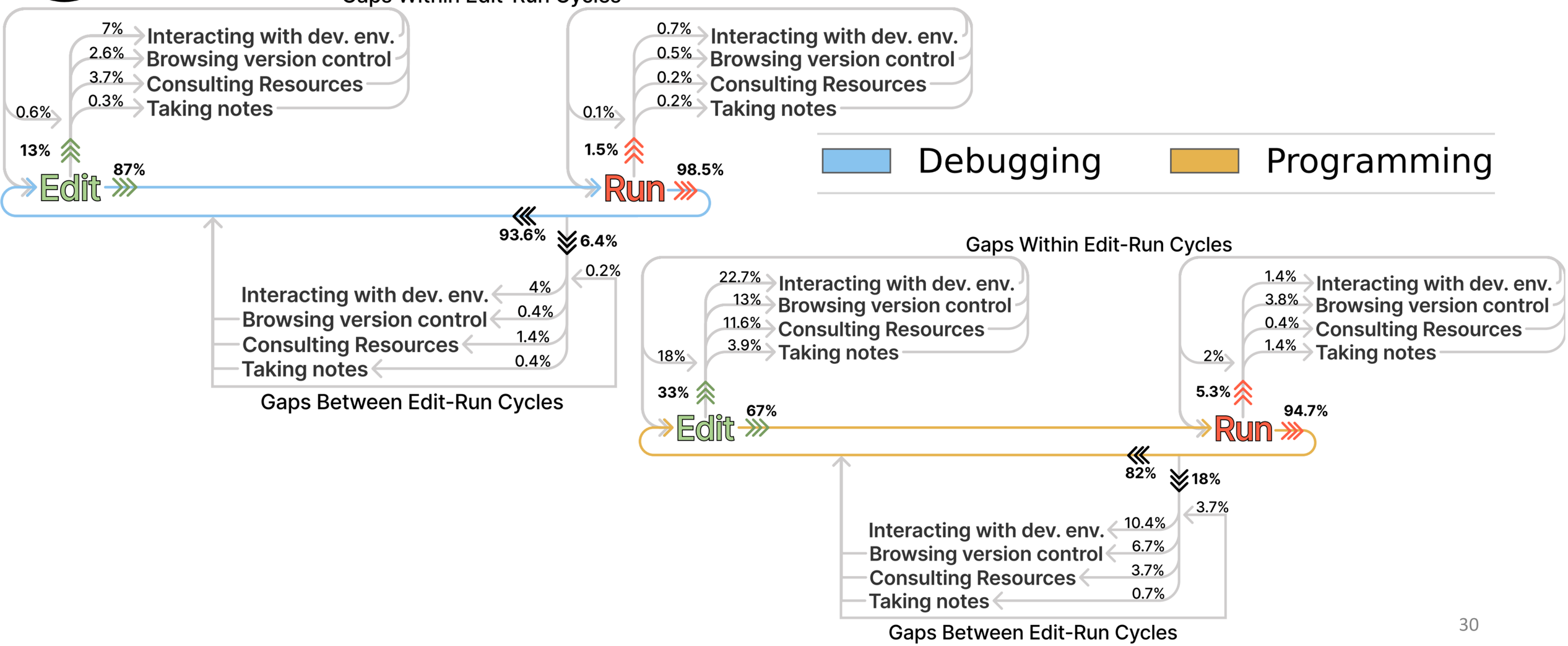


Fluidity in current edit-run cycles

RQ3: How sequential are edit-run cycles?

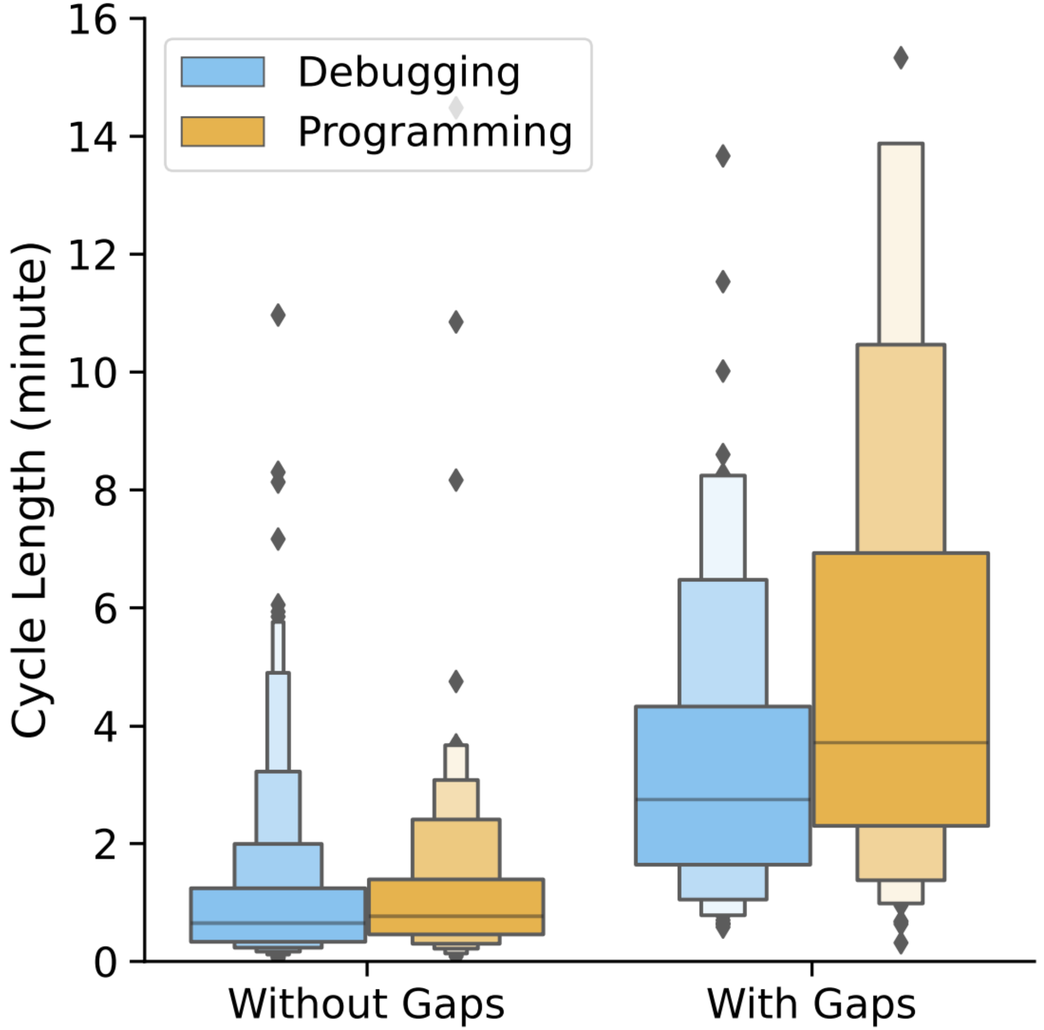
Fluidity in current edit-run cycles

RQ3: How sequential are edit-run cycles?



Fluidity in current edit-run cycles

RQ3: How sequential are edit-run cycles?

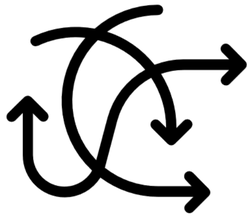


Fluidity in current edit-run cycles

RQ3: What causes gaps within and between cycles?

Fluidity in current edit-run cycles

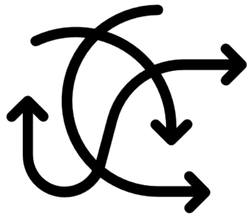
RQ3: What causes gaps within and between cycles?



Scattered Code

Fluidity in current edit-run cycles

RQ3: What causes gaps within and between cycles?

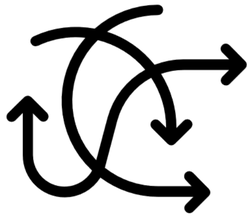


Scattered Code

Unfamiliar third-party APIs

Fluidity in current edit-run cycles

RQ3: What causes gaps within and between cycles?



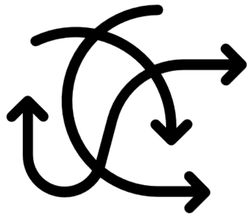
Scattered Code

Unfamiliar third-party APIs

Switch to other Tools

Fluidity in current edit-run cycles

RQ3: What causes gaps within and between cycles?



Scattered Code

Unfamiliar third-party APIs

Switch to other Tools

Waiting to compile

Fluidity in current edit-run cycles

RQ1: few minutes in length and multiple cycles in debugging and programming.

RQ2: focus on the edit step. Edit one file per cycle. Run program manually.

RQ3: Mostly sequential. However, there were needs that caused gaps between and within cycles.

What is live programming

- Programming environments that tighten the feedback loop between programming and output
- Reduce Norman's "Gulf of Evaluation" - understanding the consequences of taking an action
- Make programming more like direct manipulation, with small incremental, reversible changes with immediate feedback
- Support tinkering, exploratory programming, and learning by doing

Benefits of live programming

- minimizing the latency between a programming action and seeing its effect on program execution
- allowing performances in which programmer actions control the dynamics of the audience experience in real time
- simplifying the “credit assignment problem” faced by a programmer when some programming actions induce a new runtime behavior (such as a bug)
- supporting learning

Steven L. Tanimoto. 2013. A perspective on the evolution of live programming. In Proceedings of the 1st International Workshop on Live Programming (LIVE '13). IEEE Press, 31–34.

Demo: Learnable Programming

- http://worrydream.com/#!/LearnableProgramming

Ways to make programming more live

- Run the program whenever possible
- Show more information about the program execution

Run the program whenever possible

- Run the program whenever possible
- Quickly see what the output is

Demo: JS Bin

- https://jsbin.com/

Challenges

- Program may not be syntactically valid
- Running the program may take a long time
- Running the program may require user input

- --> easiest for small snippets
- --> edit & continue possible for larger programs

Ways to show more about execution state

- Expression values
- Data structure relationships
- Summary of function calls

Demo: Projection Boxes

- https://cseweb.ucsd.edu/~lerner/pb/

Data structure relationships

- Show runtime objects in memory and reference relationships between them
- Show how operations with collections, wrapped objects, sorts, and searches work

Demo: Python Tutor

- https://pythontutor.com/articles/java-visualizer.html

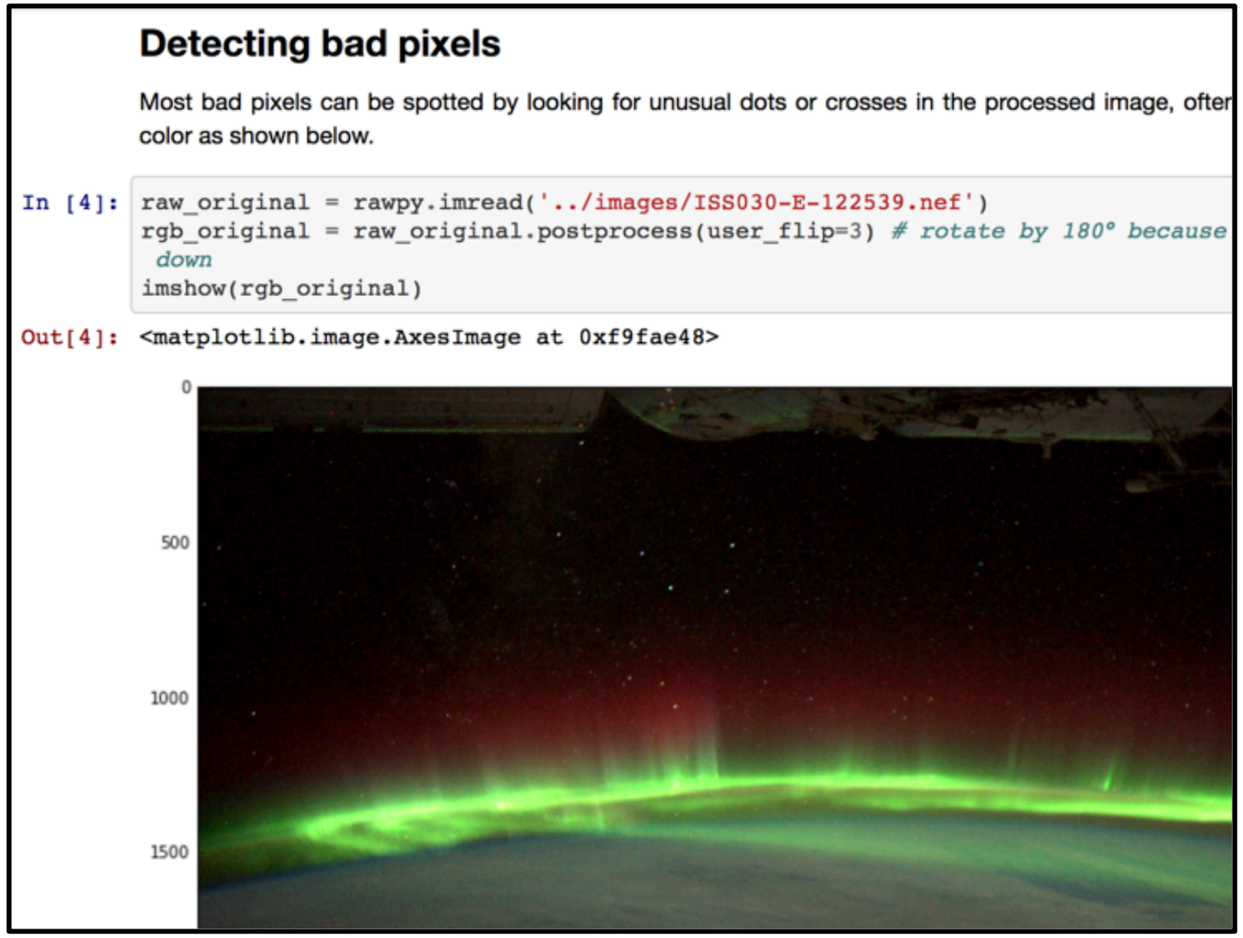
Challenges

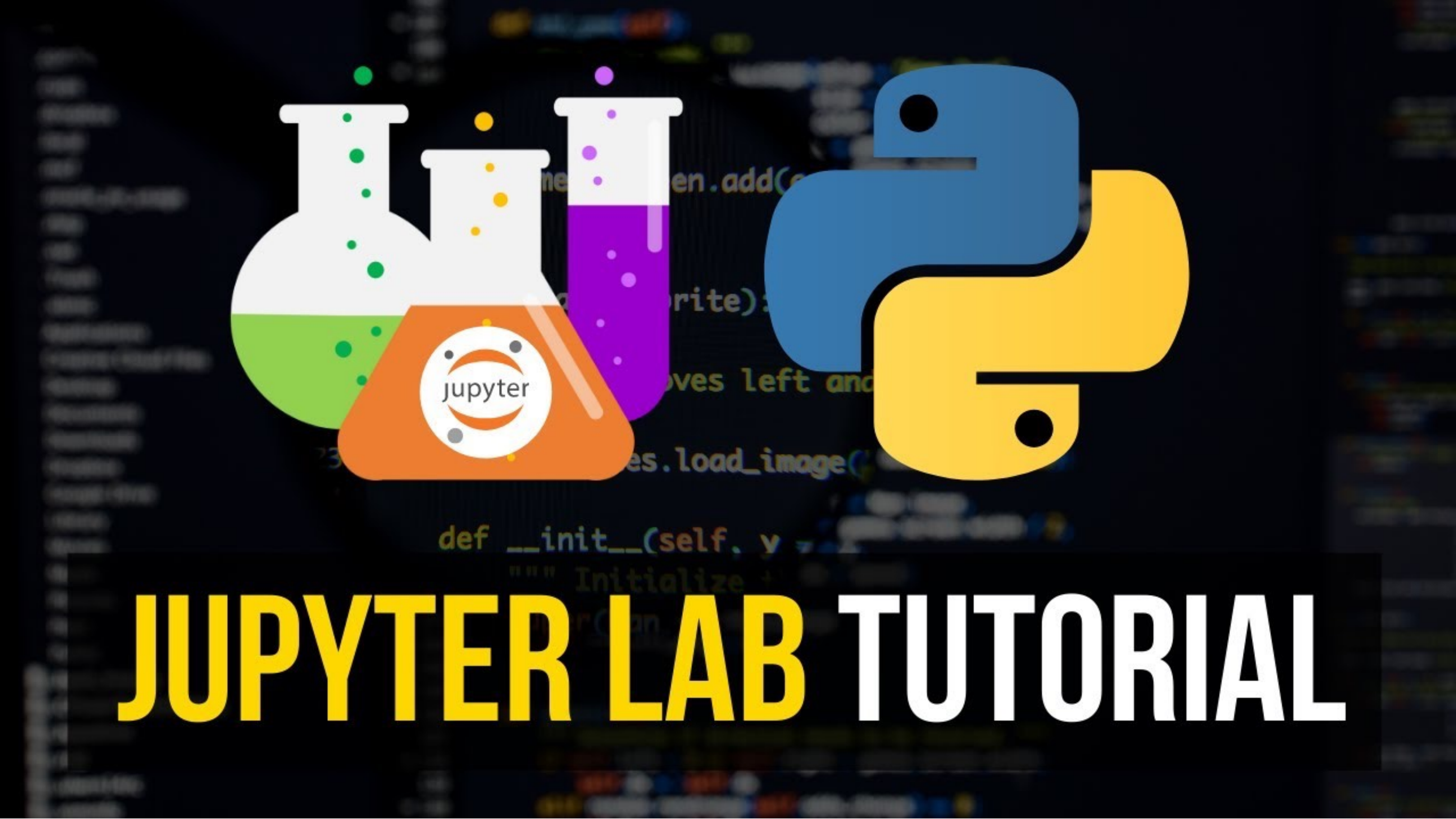
- Code executes more than once
  - How do you show the write code for
- Execution state is very, very large for real world programs
  - What to show or not show?
  - How do users find the right execution state?
- Expression values may be objects, not just primitives
  - What do you show about an object with 50 fields?

Demo: SeeCodeRun

Computational Notebooks

- Combine rich text and code to explain process of exploring and analyzing data
- Combine code and output to quickly show results of analysis scripts
- Contain cells: code, output, table, other media
- Offer execution model of running individual cells, with shared state
- Examples: Jupyter Notebooks, Mathematica, Databricks, Apache Zeppelin, Sage Notebooks




jupyter
JUPYTER LAB TUTORIAL

Use of computational notebooks

- Scratch pads - preliminary, short lived explorations to answer specific questions, debug code, test out example code
- Production pipeline - used as early version of code, to be extracted into production
- Sharing - teachers to students, computational research, data analysis

Kery, M. B., Radensky, M., Arya, M., John, B. E., & Myers, B. A. (2018). The Story in the Notebook: Exploratory Data Science using a Literate Programming Tool. In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (p. 174). Retrieved from Kery-The-Story-in-the-Notebook-Exploratory-Data-Science-using-a-Literate-Programming-Tool.pdf

Organizing notebooks

- Cells can be organized in many ways while iterating on various versions of code
  - Top to bottom - most recent last
  - Inline changes to code
- Create regions where there are multiple versions of an analysis, followed by other regions that build on previous steps
- Manage content - too many cells forces constant scrollings; distant related code cells hard to comprehend

Kery, M. B., Radensky, M., Arya, M., John, B. E., & Myers, B. A. (2018). The Story in the Notebook: Exploratory Data Science using a Literate Programming Tool. In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (p. 174). Retrieved from Kery-The-Story-in-the-Notebook-Exploratory-Data-Science-using-a-Literate-Programming-Tool.pdf

Challenges with Notebooks

Pain Point	Description	Example
Setup	Loading and cleaning data from multiple sources and platforms is a tortuous, multi-step, manual process.	"If you do a lot of data loading and pre-processing always re-loading the data is time consuming" (IP2).
Explore and Analyze	An unending cycle of copy-paste and tweaking bits of code made worse by feedback latency and kernel crashes.	"I need immediate feedback, like when I am testing slight changes in the model. I don't want to execute everything again" (IP1).
Manage Code	Managing code without software engineering support results in "dependency hell" with ad hoc workarounds that only go so far.	"Debugging is a horrible experience, copying the code over to do the debugging outside [in the IDE], and copying it back" (IP8).
Reliability	Scaling to large datasets is unsupported, causing kernel crashes and inconsistent data.	"Disconnects between browser-server or server-kernel introduce all sorts of lack-of-reliability problems" (IP6).
Archival	Preserving the history of changes and states *within* and *between* notebooks is unsupported, leading to unnecessary rework.	"The thing is using any kind of versioning mechanism for notebooks is just a complete and utter failure" (IP2).
Security	Maintaining data confidentiality and access control is an ad hoc, manual process where errors can leak private client data.	"We are missing a more private way of handling credentials. I don't want client credentials be visible to others" (IP13).
Share and Collaborate	Sharing data or parts of the notebook interactively and at different levels—demo/reports, review/comment, collaborative editing—is generally unsupported.	"There are cases where somebody is asking you to review/comment, while other times to go collaborate" (IP6).
Reproduce and Reuse	Replicating results or reusing parts of code is infeasible because of high levels of customization and environment dependencies.	"The fact that somebody could run a notebook on organization A's service but not on organization B's is a serious problem" (IP6).
Notebooks as Products	Deploying to production requires significant cleanup and packaging of libraries—DevOps skills that are outside the core skill set of data scientists.	"Once the code gets a certain level of maturity, it's very difficult to transition that to production code. Everything has to translate to functions and classes" (IP15).

Souti Chattopadhyay, Ishita Prasad, Austin Z. Henley, Anita Sarma, and Titus Barik. 2020. What's Wrong with Computational Notebooks? Pain Points, Needs, and Design Opportunities. In Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems (CHI '20). Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3313831.3376729

10 min break

Tech Talks

In-Class Activity

- In groups of 2 or 3, try out CodePen.io or SeeCode.run
  - Build a simple calculator (e.g., buttons to add, subtract, multiple, delete)
  - Reflect on your experiences with live programming tool
    - What were you able to accomplish (totally ok if didn't finish)
    - What worked well
    - What didn't work well

- Submission
  - Submit (1) pdf or doc with reflection and (2) source code through Blackboard. 1 submission per group. Due 7pm today.