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
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

Edit

Run

Editing a file of code
Browsing a file of code

Testing Program
Inspecting Program
Observe-dev dataset

Activities

Edit
Editing a file of code
Browsing a file of code

Run
Testing Program
Inspecting Program

Gaps
Consulting Resources
Others
Observe-dev dataset

Activities

- Browsing a file of code
- Editing 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?

- 1 minutes long (Debugging)
- 3 minutes long (Programming)
**Fluidity in current edit-run cycles**

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

1 minute long (Debugging)
3 minutes long (Programming)

7 cycles (Debugging)
2 cycles (Programming)
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?
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

- Edit and Run cycles distribution
- Manual vs. Tests
- Files Edited: 0, 1, 2, +3
- Output: Final, States

Cycles Length: 0% to 100%
Edit-Run Cycles: 0% to 100%
Fluidity in current edit-run cycles

**RQ3:** How sequential are edit-run cycles?
**RQ3:** How sequential are edit-run cycles?

Fluidity in current edit-run cycles

Gaps Within Edit-Run Cycles

- Interacting with dev. env.: 7%
- Browsing version control: 2.6%
- Consulting Resources: 3.7%
- Taking notes: 0.3%

Gaps Between Edit-Run Cycles

- Interacting with dev. env.: 4%
- Browsing version control: 0.4%
- Consulting Resources: 1.4%
- Taking notes: 0.4%

Gaps Within Edit-Run Cycles

- Interacting with dev. env.: 0.7%
- Browsing version control: 0.5%
- Consulting Resources: 0.2%
- Taking notes: 0.2%

Gaps Between Edit-Run Cycles

- Interacting with dev. env.: 10.4%
- Browsing version control: 6.7%
- Consulting Resources: 3.7%
- Taking notes: 0.7%
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
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

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/](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 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

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

# Challenges with Notebooks

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

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.