Debugging Support for Reactive Programming

Supplementary Material

Anonymous Author(s)

Contents

Introduction

This document complements the research paper "Debugging Support for Reactive Programming: Feasibility of a Ready-to-hand Debugger for RxJS" during the double-blind review process.

Cognitive Walkthrough

The cognitive walkthrough report formally follows the guide by Wharton et al. [2]. Further, the report refers to "operator log points" as "probes." This is because we called the log point concept differently during the proof of concept phase and later transitioned to the more intuitive name, based on usability test results.

Persona "Frank Flow"

Profile

- Age: 29 years
- Gender: Male
- Education: BSc in Computer Science
- Occupation: Frontend Software Engineer at ReactiBank

Frank started to work for ReactiBank 2 years ago as a frontend software engineer. As part of a small, interdisciplinary team of 7 people, Frank' and his team are responsible for developing and maintaining a trading application. This application relies heavily on real-time data, so the group decided to use reactive programming principles throughout the application. Frank knows traditional programming paradigms and the related debugging tools from his studies and personal experiences. He built up knowledge on RP and RxJS for the frontend part of their application after joining the team quickly, however.

Today, Frank uses RxJS efficiently to build new features. He can solve simple problems reported by the product owner on his own. Working on more complicated issues is still something Frank struggles with: He often feels like his knowledge of traditional programming techniques and its debugging utilities are not enough. These tools feel "out of place" to him and do not provide the answers he is looking for. Frank does not like that, eventually, he has to consult one of his colleagues who have experience in RxJS for a longer time.

Goals

- Make complex business domains simple and easy to use for everyone
- Build beautiful, responsive and easy-to-use user interfaces
- Be a fully productive member of the team
- Understand RxJS in complex setups better and deepen knowledge on it

Frustrations

- Known debugging utilities seem unfit to provide answers regarding RP code

Setup

Context

This cognitive walkthrough is based on the first problem given to subjects during the observational study of Alabor et al. [1].

User

See Section "Persona Frank Flow".

Task

After I started the "Problem 1" application and inspected its UI, I was able to observe multiple, unexpected updates rendered in quick succession after I clicked the reset button. Based on this evidence, I formulate my first debugging hypothesis: I suspect that the `flatMap` operator on Line 18 in the file `index.ts` does create multiple observables, which do not get unsubscribed when the reset button is clicked. This results in the observed behavior eventually. To proof my hypothesis, I want to inspect the life cycle events of the created observables more closely.

Environment

Visual Studio Code with enabled TypeScript support is installed. The prototype of the RxJS debugging extension is installed as well. The source code of "Problem 1" [1] is present. Further, an internet browser (e.g. Mozilla Firefox or Google Chrome) is present.

Walkthrough

Open File

Open `index.ts` in Visual Studio Code.

- Visual Studio Code: Shows contents of `index.ts` file.
- Success story:
  - We can expect the user to open `index.ts` since he already suspects a problem within this file as stated in the original task.

Navigate to Operator

Move cursor the `flatMap` operator on Line 18.

- Visual Studio Code: Shows code actions icon in front of Line 18.
- Success story:

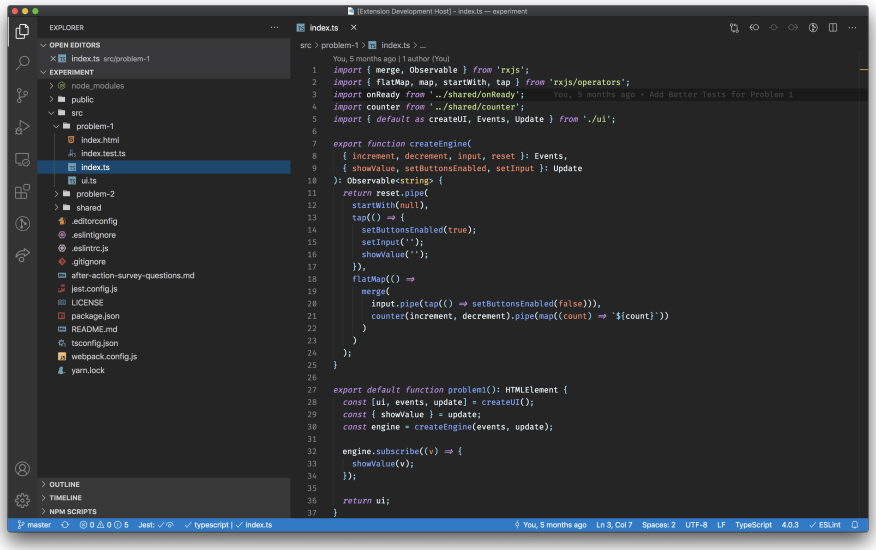


Figure 1: Visual Studio Code after opening the `index.ts` file.

  - The original task clearly describes the hypothesis regarding this line/piece of source code. Hence, navigating here seems the natural course of action for the user.

Open Code Actions

Open the code actions menu by clicking the yellow light bulb icon.

- Visual Studio Code: Shows available code actions.
- Failure story:
  - Will the user know that the correct action is available?
    * The user might know code actions for providing options to refactor a piece of code or quick fixes for code linting problems. It is questionable if he will expect functionality to inspect parts of a data flow graph here.

Create Probe for Operator

Select "Probe Observable..." code action from the related menu.

- Visual Studio Code: Adds `flatMap` operator on Line 18 to "Observables" list in debugging view.
- Failure story:

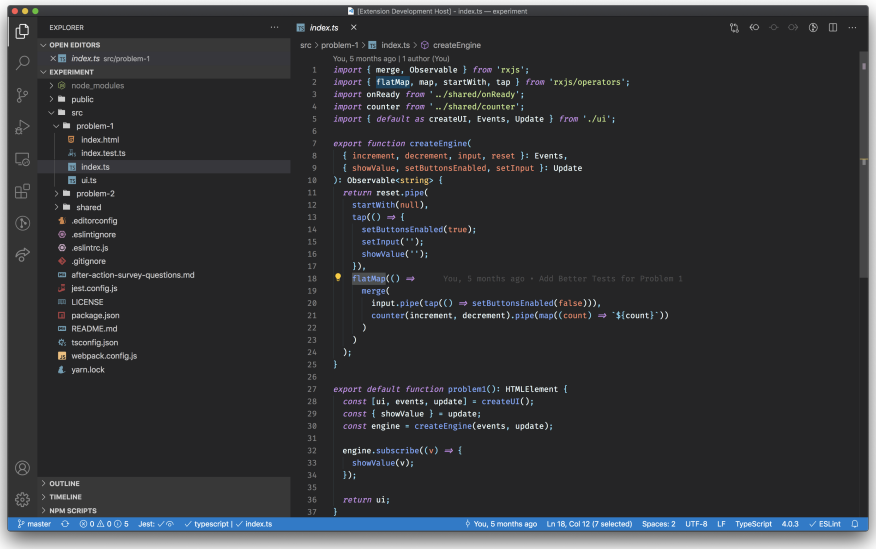


Figure 2: Visual Studio Code after navigating cursor to the `flatMap` operator on Line 18.

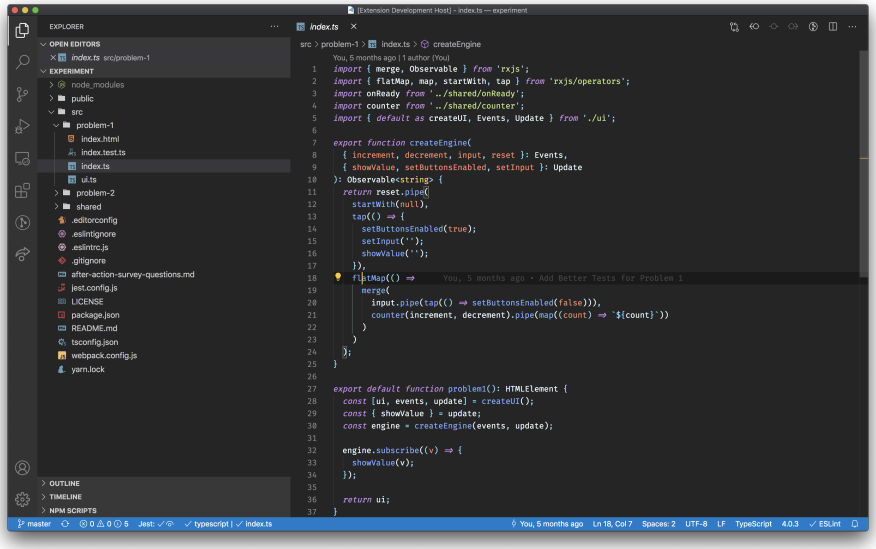


Figure 3: Visual Studio Code indicating available code actions on Line 18 using a yellow light bulb icon.

  - If the correct action is taken, will the user see that things are going ok?
    * The “Observables” list is part of the debugging view of Visual Studio Code. The user will not get any feedback that his action “Probe Observable...” was successful without changing the view manually to debugging and expanding the “Observables” panel in the lower left.

Open Observable Probe Monitor

Open the “Observable Probe Monitor” view using command palette.

- Visual Studio Code: Shows empty “Observable Probe Monitor” view
- Failure story:
  - Will the user know that the correct action is available?
    * The user might not be aware that the “Observable Probe Monitor” view is hidden within the command palette. Hence, they might feel lost after adding the observable probe in the previous step.
  - If the correct action is taken, will the user see that things are going ok?
    * The user might get confused by the “Observable Probe Monitor” being blank by default.

Launch Application

Execute “Problem 1” launch configuration

- Visual Studio Code: Opens default browser showing “Problem 1”
- Default Browser: Shows “Problem 1” UI
- Success story:
  - The users previous experience with Visual Studio Code launch configuration allows assuming this the natural course of action in order to prepare himself for further inspection of the application.

Interact with Application

Interact with “Problem 1” in the default browser.

- Visual Studio Code: “Observable Probe Monitor” provides live telemetry information about values and life cycle events produced by the `flatMap` operator.
- Failure story:

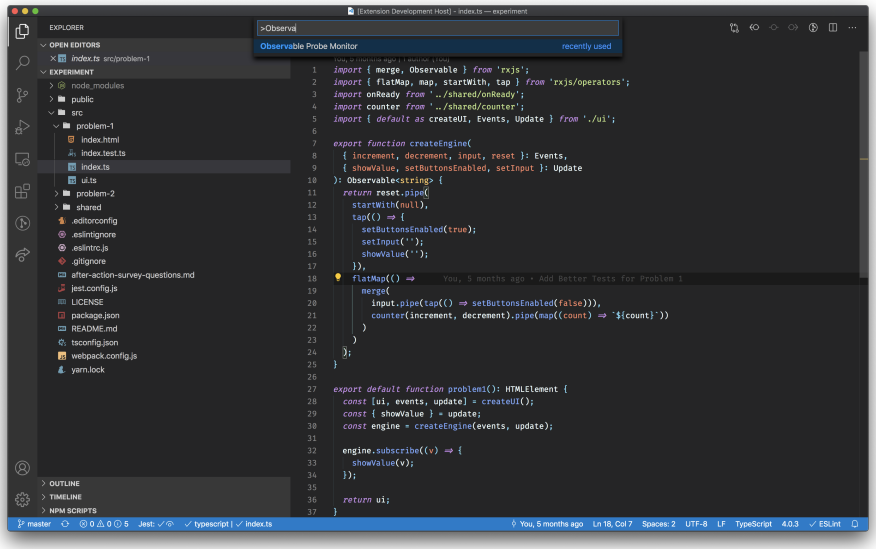


Figure 4: Visual Studio Codes command palette menu showing the "Observable Probe Monitor" command.

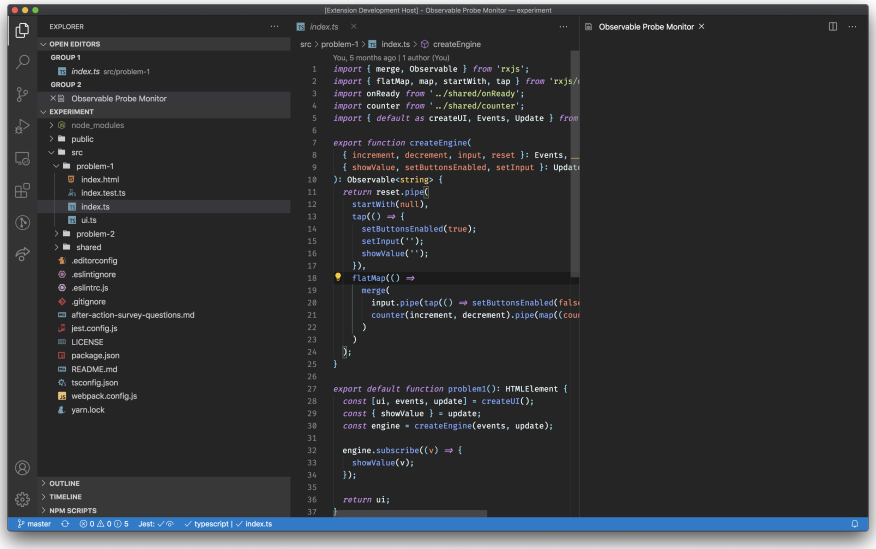


Figure 5: Visual Studio Code showing the empty Observable Probe Monitor on the right pane.

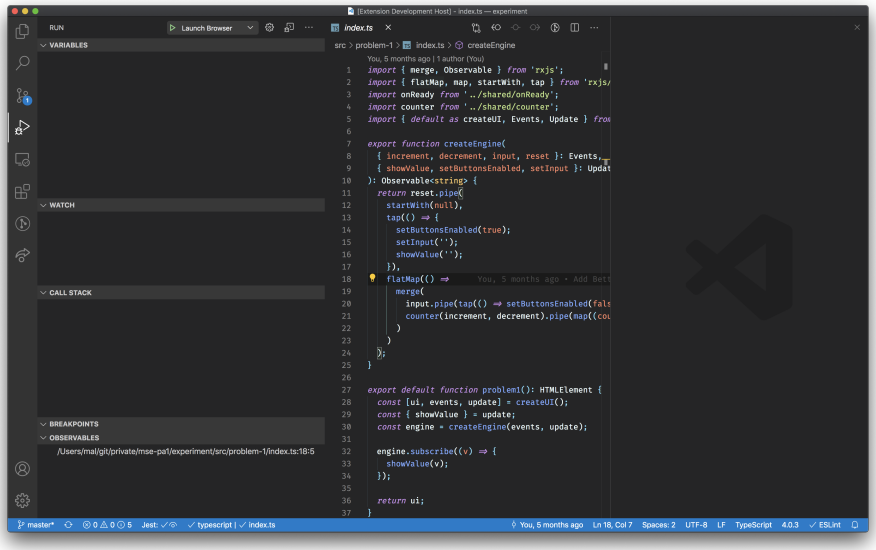


Figure 6: Visual Studio Code showing the debugging view after launching "Problem 1."

- Will the user know that the correct action will achieve the desired effect?
  - The user might not be aware that he is expected to interact with "Problem 1" in the default browser in order to get live feedback in the "Observable Probe Monitor."
- If the correct action is taken, will the user see that things are going ok?
  - The default browser might overlay Visual Studio Code and the "Observable Probe Monitor" view. This is why the user might miss the live trace of values and life cycle events displayed in the "Observable Probe Monitor."

Interpret Runtime Behavior

Interpret the live trace of emitted values and life cycle events in the "Observable Probe Monitor" view

- Visual Studio Code: Provides detail information to a traced item
- Success story:
  - The original task states that the user is interested in more close information regarding the `flatMap` operator. Since the "Observable

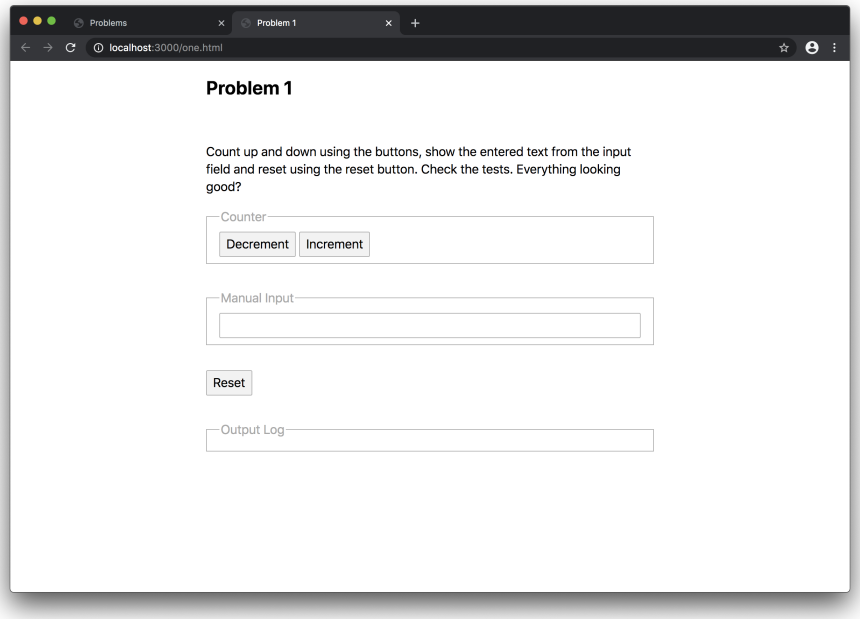


Figure 7: Google Chrome displaying the user interface of “Problem 1” ready to receive interactions.

Probe Monitor" provide such information in real-time, we can expect the user to use this information accordingly.

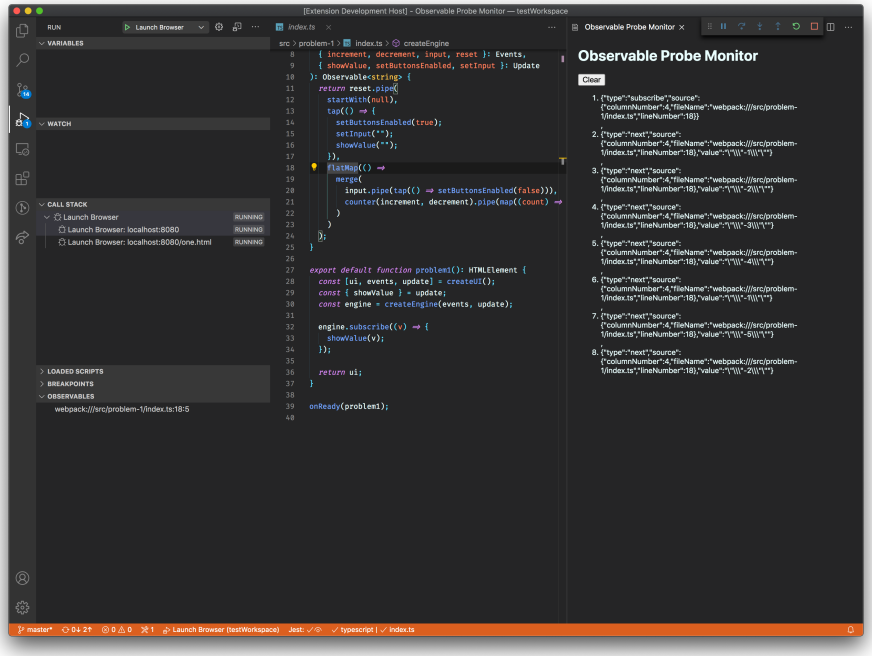


Figure 8: Visual Studio Code showing live telemetry in the "Observable Probe Monitor."

Failure Stories

This is a summary of all failure stories identified during the cognitive walkthrough.

Step	Failure Story
Open Code Actions	The user might know code actions for providing options to refactor a piece of code or quick fixes for code linting problems. It is questionable if he will expect functionality to inspect parts of a data flow graph here.
Create Probe for Operator	The “Observables” list is part of the debugging view of Visual Studio Code. The user will not get any feedback that his action “Probe Observable...” was successful without changing the view manually to debugging and expanding the “Observables” panel in the lower left.
Open Observable Probe Monitor	The user might not be aware that the “Observable Probe Monitor” view is hidden within the command palette. Hence, they might feel lost after adding the observable probe in the previous step.
Open Observable Probe Monitor	The user might get confused by the “Observable Probe Monitor” being blank by default.
Interact with Application	The user might not be aware that he is expected to interact with “Problem 1” in the default browser in order to get live feedback in the “Observable Probe Monitor.”
Interact with Application	The default browser might overlay Visual Studio Code and the “Observable Probe Monitor” view. This is why the user might miss the live trace of values and life cycle events displayed in the “Observable Probe Monitor.”

Usability Test

Observed Issues

These are all usability issues identified during the usability test sessions.

Participant(s)	Phase	Task	Problem
P2, P3	Instrument Hypothesis	Environment Setup	Subject starts the application in debugging mode, even though they have started it before already.
P2, P3	Instrument Hypothesis	Manage Log Points	Subject unable to find log point list in debugging view.
P2	Instrument Hypothesis	Manage Log Points	Subject unable to identify already defined log points.
P2	Instrument Hypothesis	Interpret Log	Subject cannot find “Clear” button to clear the log before starting a new debugging iteration.
P3	Instrument Hypothesis	Manage Log Points	Subject cannot add log point to an observable.
P3	Instrument Hypothesis	Manage Log Points	Subject cannot add log point by clicking the editors gutter.*(Regular break points are added here)*
P2, P3	Test Hypothesis	Interpret Log	Subject has difficulties to make a connection from a log point to the generated log entry.
P2, P3	Test Hypothesis	Interpret Log	Subject interprets logged value as the “input” of the instrumented operator.
P2	Test Hypothesis	Interpret Log	Subject is overwhelmed by multiple log entries generated by multiple log points.
P3	Test Hypothesis	Interpret Log	Subject does not see log entries when running the unit test suite.

Marketplace Presence

The next page shows an excerpt of the Visual Studio Marketplace presence of the “RxJS Debugging for Visual Studio Code” extension as of 2021-12-23.

RxJS Debugging for Visual Studio Code

[redacted] | 922 installs | ★★★★★ (1) | Free

Add non-intrusive debugging capabilities for RxJS applications to Visual Studio Code.

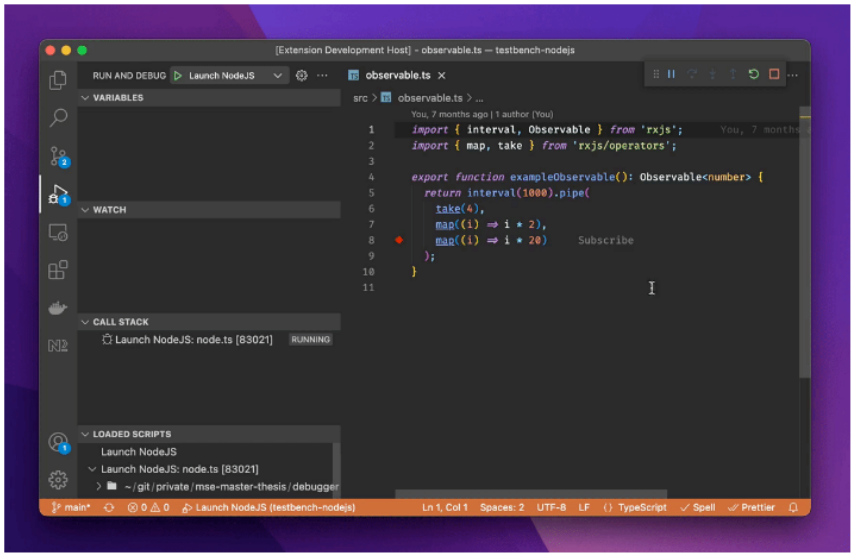
Install Trouble Installing?

RxJS Debugging for Visual Studio Code

VS Marketplace v1.1.1 Follow @rxjsdebuggung

> Never, ever use `tap(console.log)` again.

Add non-intrusive debugging capabilities for RxJS applications to Visual Studio Code.



Features

- RxJS debugging, fully integrated with Visual Studio Code
- Works with RxJS 6.6.7 and newer
- Support for Node.js and Webpack-based RxJS applications

Requirements

- Visual Studio Code 1.61 or newer
- RxJS 6.6.7 or newer
- To debug NodeJS-based applications:
  - Node.js 12 or newer
- To debug Webpack-based web applications:
  - Webpack 5.60.0 or newer
  - The latest @rxjs-debugging/runtime-webpack Webpack plugin (see here for setup instructions)

Categories

Debuggers

Tags

javascript javascriptreact typescript typescriptreact

Works with

Universal

Resources

Issues
Repository
Homepage
License
Changelog
Download Extension

Project Details

[redacted]
Last Commit: 2 weeks ago
11 Pull Requests
19 Open Issues

More Info

Version	1.1.1
Released on	17/05/2021, 15:58:57
Last updated	08/12/2021, 15:30:44
Publisher	[redacted]
Unique Identifier	[redacted]
Report	Report Abuse

References

[1] Manuel Alabor and Markus Stolze. 2020. Debugging of RxJS-based applications. In *Proceedings of the 7th ACM SIGPLAN international workshop on reactive and event-based languages and systems* (REBLS 2020), Association for Computing Machinery, 15–24. DOI:https://doi.org/10.1145/3427763.3428313

[2] Cathleen Wharton, John Rieman, Lewis Clayton, and Peter Polson. 1994. *The cognitive walkthrough: A practitioner's guide.* Institute of Cognitive Science, University of Colorado.