User Journey: Debugging of RxJS-Based Applications

Manuel Alabor
Eastern Switzerland University of Applied Sciences
manuel.alabor@ost.ch

This document describes two user journeys of Frank Flow, a software engineer working on an application implemented using RxJS. The application has a web-based user interface and was part of a study by Alabor et al. [1]. Frank will use different debugging techniques and utilities to solve the task given to him. The journeys follow the debugging process model proposed by Layman et al. [2].

Actor

Frank Flow 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. All to often, they do not provide any helpful insights and turn out to be time-sink. Frank does not like that, eventually, he has to consult one of his colleagues who have experience in RxJS for a longer time.

Context

Frank participates in a study about RxJS-specific debugging techniques. He is asked to work on an application allowing a user to change and display a value. The user can achieve a change of the value either by two buttons to increase and decrease a numeric value or by entering an arbitrary text in an input field. The two buttons get disabled once a text is entered. There is a third button labeled "Reset", which clears all user input and reverts the application to its initial state.

Goal

Frank receives a bug report for the application: Once a user had clicked the reset button, the application started to behave strangely by showing multiple values in quick succession when the increase or decrease button was clicked. It is Franks task to analyse the bug and find a solution.

User Journey

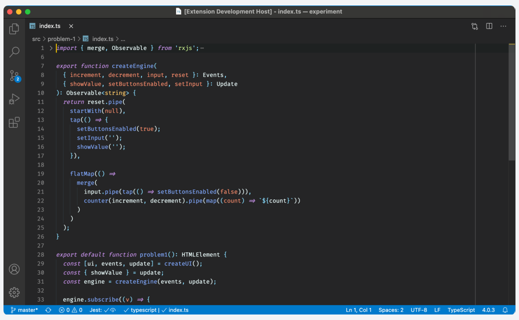
Beginning from the same point, we are going to present two distinct user journeys in which Frank uses different tools to achieve his goal:

Branch A on the left describes a debugging session using traditional debugging techniques and practices as described by Alabor et al. [1]. Frank will use imperative debugging utilities built-in to Visual Studio Code and manual code modification to reach his goal.
Branch B on the right describes how Frank uses the prototype of an extension for Visual Studio Code providing a less invasive debugging technique for RxJS-based source code.

Step 1 **Build Hypothesis**

After Frank recreated the behavior reported in the bug report, he starts analyzing the source code of the application. In this process, he formulates his first hypothesis about what could cause the unexpected behavior: Frank suspects the `flatMap` operator in Line 19 to be responsible.

Frank wants to have a closer look on the operators runtime behavior, which is why he decides to use debugging utilities.

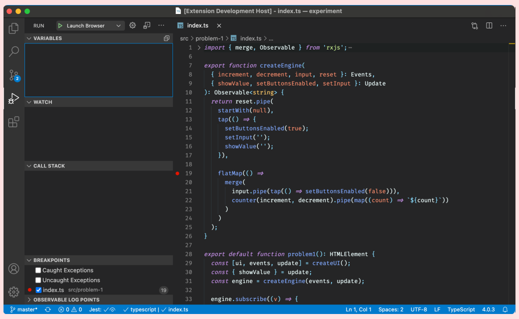

[Extension Development Host] - index.ts — experiment
index.ts
src > problem-1 > index.ts > ...

```
 1 > import { merge, Observable } from 'rxjs';⋯
 6
 7   export function createEngine(
 8     { increment, decrement, input, reset }: Events,
 9     { showValue, setButtonsEnabled, setInput }: Update
10   ): Observable<string> {
11     return reset.pipe(
12       startWith(null),
13       tap(() => {
14         setButtonsEnabled(true);
15         setInput('');
16         showValue('');
17       }),
18
19       flatMap(() =>
20         merge(
21           input.pipe(tap(() => setButtonsEnabled(false))),
22           counter(increment, decrement).pipe(map((count) => `${count}`))
23         )
24       )
25     );
26   }
27
28   export default function problem1(): HTMLElement {
29     const [ui, events, update] = createUI();
30     const { showValue } = update;
31     const engine = createEngine(events, update);
32
33     engine.subscribe((v) => {
```

master* ⊗ 0 ⚠ 0 Jest: ✓ ✓ typescript | ✓ index.ts
Ln 1, Col 1 Spaces: 2 UTF-8 LF TypeScript 4.0.3

Step 2.A **Instrument Hypothesis**

Using the built-in debugger of Visual Studio Code, Frank adds a breakpoint to Line 19 where the `flatMap` operator is called with the intention to stop the program execution every time a values "flows" through the operator.



[Extension Development Host] - index.ts — experiment

RUN ▷ Launch Browser | index.ts

src > problem-1 > index.ts > ...

```
 1 > import { merge, Observable } from 'rxjs';…
 6
 7   export function createEngine(
 8     { increment, decrement, input, reset }: Events,
 9     { showValue, setButtonsEnabled, setInput }: Update
10   ): Observable<string> {
11     return reset.pipe(
12       startWith(null),
13       tap(() => {
14         setButtonsEnabled(true);
15         setInput('');
16         showValue('');
17       }),
18
19 ●     flatMap(() =>
20         merge(
21           input.pipe(tap(() => setButtonsEnabled(false))),
22           counter(increment, decrement).pipe(map((count) => `${count}`))
23         )
24       )
25     );
26   }
27
28   export default function problem1(): HTMLElement {
29     const [ui, events, update] = createUI();
30     const { showValue } = update;
31     const engine = createEngine(events, update);
32
33     engine.subscribe((v) => {
```

VARIABLES

WATCH

CALL STACK

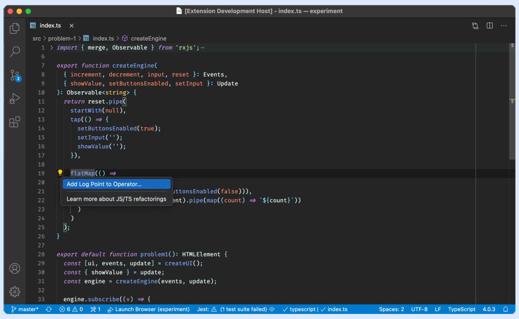
BREAKPOINTS
- Caught Exceptions
- Uncaught Exceptions
- ✓ index.ts src/problem-1 19

OBSERVABLE LOG POINTS

master* ⊗ 0 ⚠ 0 Jest: ✓ typescript | ✓ index.ts Ln 1, Col 1 Spaces: 2 UTF-8 LF TypeScript 4.0.3

Step 2.B **Instrument Hypothesis**

Frank navigates to the `flatMap` operator in Line 19 and selects the "Add Log Point to Operator..." code action provided by the Visual Studio Code extension.



[Extension Development Host] - index.ts — experiment

index.ts

src > problem-1 > index.ts > createEngine

```
 1 > import { merge, Observable } from 'rxjs';…
 6
 7   export function createEngine(
 8     { increment, decrement, input, reset }: Events,
 9     { showValue, setButtonsEnabled, setInput }: Update
10   ): Observable<string> {
11     return reset.pipe(
12       startWith(null),
13       tap(() => {
14         setButtonsEnabled(true);
15         setInput('');
16         showValue('');
17       }),
18
19       flatMap(() =>
20
21                           uttonsEnabled(false))),
22                           ent).pipe(map((count) => `${count}`))
23         )
24       )
25     );
26   }
27
28   export default function problem1(): HTMLElement {
29     const [ui, events, update] = createUI();
30     const { showValue } = update;
31     const engine = createEngine(events, update);
32
33     engine.subscribe((v) => {
```

Add Log Point to Operator...
Learn more about JS/TS refactorings

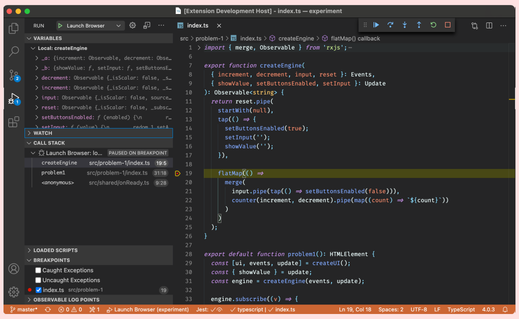
master* ⊗ 6 ⚠ 0 1 Launch Browser (experiment) Jest: ⚠ (1 test suite failed) ✓ typescript | ✓ index.ts Spaces: 2 UTF-8 LF TypeScript 4.0.3

Step 3.A **Test Hypothesis**

Frank launches the application. Before the web browser can display anything, the debugger in Visual Studio Code halts the program execution at the breakpoint in Line 19.

Contrary to Franks expectation, the application was already paused during the creation of the `flatMap` operator rather than when a value was processed by it.

Frank resumes the program execution, though the breakpoint never pauses the application again.



[Extension Development Host] - index.ts — experiment

RUN ▷ Launch Browser | index.ts

src > problem-1 > index.ts > createEngine > flatMap() callback

VARIABLES
- Local: createEngine
  - > _a: {increment: Observable, decrement: Obse…
  - > _b: {showValue: f, setInput: f, setButtonsE…
  - > decrement: Observable {_isScalar: false, _s…
  - > increment: Observable {_isScalar: false, _s…
  - > input: Observable {_isScalar: false, source…
  - > reset: Observable {_isScalar: false, _subsc…
  - > setButtonsEnabled: f (enabled) {\n r…
  - > setInput: f (value) {\n redom_1.setA…

WATCH

CALL STACK
- Launch Browser: lo… PAUSED ON BREAKPOINT
  - createEngine src/problem-1/index.ts 19:5
  - problem1 src/problem-1/index.ts 31:18
  - <anonymous> src/shared/onReady.ts 9:28

```
 1 > import { merge, Observable } from 'rxjs';…
 6
 7   export function createEngine(
 8     { increment, decrement, input, reset }: Events,
 9     { showValue, setButtonsEnabled, setInput }: Update
10   ): Observable<string> {
11     return reset.pipe(
12       startWith(null),
13       tap(() => {
14         setButtonsEnabled(true);
15         setInput('');
16         showValue('');
17       }),
18
19       flatMap(() =>
20         merge(
21           input.pipe(tap(() => setButtonsEnabled(false))),
22           counter(increment, decrement).pipe(map((count) => `${count}`))
23         )
24       )
25     );
26   }
27
28   export default function problem1(): HTMLElement {
29     const [ui, events, update] = createUI();
30     const { showValue } = update;
31     const engine = createEngine(events, update);
32
33     engine.subscribe((v) => {
```

LOADED SCRIPTS

BREAKPOINTS
- Caught Exceptions
- Uncaught Exceptions
- ✓ index.ts src/problem-1 19

OBSERVABLE LOG POINTS

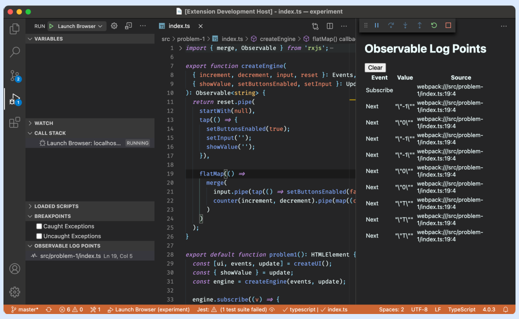
master* ⊗ 0 ⚠ 0 1 Launch Browser (experiment) Jest: ✓ typescript | ✓ index.ts Ln 19, Col 18 Spaces: 2 UTF-8 LF TypeScript 4.0.3

Step 3.B **Test Hypothesis**

Frank launches the application. While reproducing the reported bug in the browser, the extension produces a log of all events detected at the `flatMap` operator in Line 19.

Frank recognizes a peculiar pattern: After the reset button was clicked, the log point reports multiple values emitted for each click on the increase/decrease button.

This confirms Franks hypothesis about the `flatMap` operator: After the user clicked the reset button, the operator emits multiple values.



[Extension Development Host] - index.ts — experiment

RUN ▷ Launch Browser | index.ts

src > problem-1 > index.ts > createEngine > flatMap() callba

```
 1 > import { merge, Observable } from 'rxjs';…
 6
 7   export function createEngine(
 8     { increment, decrement, input, reset }: Events,
 9     { showValue, setButtonsEnabled, setInput }: Upd
10   ): Observable<string> {
11     return reset.pipe(
12       startWith(null),
13       tap(() => {
14         setButtonsEnabled(true);
15         setInput('');
16         showValue('');
17       }),
18
19       flatMap(() =>
20         merge(
21           input.pipe(tap(() => setButtonsEnabled(fa
22           counter(increment, decrement).pipe(map((c
23         )
24       )
25     );
26   }
27
28   export default function problem1(): HTMLElement {
29     const [ui, events, update] = createUI();
30     const { showValue } = update;
31     const engine = createEngine(events, update);
32
33     engine.subscribe((v) => {
```

VARIABLES

WATCH

CALL STACK
- Launch Browser: localhos… RUNNING

LOADED SCRIPTS

BREAKPOINTS
- Caught Exceptions
- Uncaught Exceptions

OBSERVABLE LOG POINTS
- src/problem-1/index.ts Ln 19, Col 5

Observable Log Points

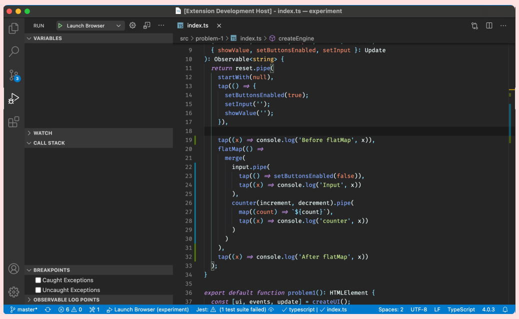
Clear

Event	Value	Source
Subscribe		webpack:///src/problem-1/index.ts:19:4
Next	"\"-1\""	webpack:///src/problem-1/index.ts:19:4
Next	"\"0\""	webpack:///src/problem-1/index.ts:19:4
Next	"\"-1\""	webpack:///src/problem-1/index.ts:19:4
Next	"\"-1\""	webpack:///src/problem-1/index.ts:19:4
Next	"\"0\""	webpack:///src/problem-1/index.ts:19:4
Next	"\"0\""	webpack:///src/problem-1/index.ts:19:4
Next	"\"1\""	webpack:///src/problem-1/index.ts:19:4
Next	"\"1\""	webpack:///src/problem-1/index.ts:19:4
Next	"\"1\""	webpack:///src/problem-1/index.ts:19:4

master* ⊗ 6 ⚠ 0 1 Launch Browser (experiment) Jest: ⚠ (1 test suite failed) ✓ typescript | ✓ index.ts Spaces: 2 UTF-8 LF TypeScript 4.0.3

Step 4.A **Instrument Hypothesis**

After Franks first failed instrumentation attempt using breakpoints, he decides to add trace log statements using the `tap` operator manually. He adds multiple of them on Lines 19, 24, 28, and 32.

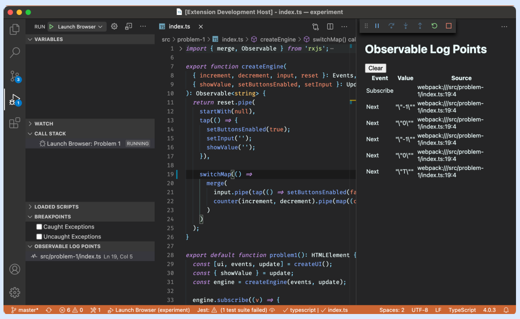


```
 9    { showValue, setButtonsEnabled, setInput }: Update
10  ): Observable<string> {
11    return reset.pipe(
12      startWith(null),
13      tap(() => {
14        setButtonsEnabled(true);
15        setInput('');
16        showValue('');
17      }),
18
19      tap((x) => console.log('Before flatMap', x)),
20      flatMap(() =>
21        merge(
22          input.pipe(
23            tap(() => setButtonsEnabled(false)),
24            tap((x) => console.log('Input', x))
25          ),
26          counter(increment, decrement).pipe(
27            map((count) => `${count}`),
28            tap((x) => console.log('counter', x))
29          )
30        )
31      ),
32      tap((x) => console.log('After flatMap', x))
33    );
34  }
35
36  export default function problem1(): HTMLElement {
37    const [ui, events, update] = createUI();
```

Step 4.B **Resolve Bug**

After researching the RxJS documentation, Frank realizes that the `flatMap` operator [3] does not unsubscribe observables it created earlier. This sounds like a reasonable explanation for the observed behavior.

Frank replaces the faulty operator with the `switchMap` [4] operator.



```
import { merge, Observable } from 'rxjs';

export function createEngine(
  { increment, decrement, input, reset }: Events,
  { showValue, setButtonsEnabled, setInput }: Upd
): Observable<string> {
  return reset.pipe(
    startWith(null),
    tap(() => {
      setButtonsEnabled(true);
      setInput('');
      showValue('');
    }),

    switchMap(() =>
      merge(
        input.pipe(tap(() => setButtonsEnabled(fa
        counter(increment, decrement).pipe(map((c
      )
    )
  );
}

export default function problem1(): HTMLElement {
  const [ui, events, update] = createUI();
  const { showValue } = update;
  const engine = createEngine(events, update);

  engine.subscribe((v) => {
```

Observable Log Points

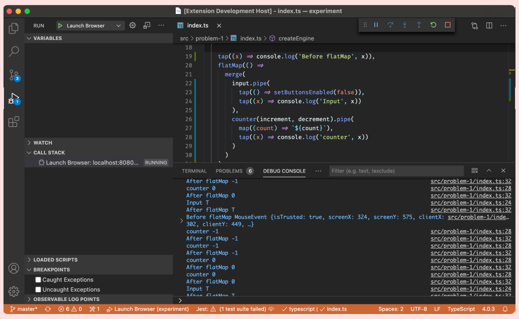
Event	Value	Source
Subscribe		webpack:///src/problem-1/index.ts:19:4
Next	"\"-1\""	webpack:///src/problem-1/index.ts:19:4
Next	"\"0\""	webpack:///src/problem-1/index.ts:19:4
Next	"\"-1\""	webpack:///src/problem-1/index.ts:19:4
Next	"\"0\""	webpack:///src/problem-1/index.ts:19:4
Next	"\"T\""	webpack:///src/problem-1/index.ts:19:4

Step 5.A **Test Hypothesis**

Frank launches the application again. The manually added code generates the expected trace log in the debuggers console as Frank executes the steps necessary to recreate the reported bug.

Once the application produced enough logs, Frank starts to analyse them. Even though it is hard to reassign a log entry to a piece of code and a related action, Frank recognizes a peculiar pattern after some time: After the reset button was clicked, the log statement on Line 32 is executed multiple times, even though the increase/decrease button gets clicked only once.

This confirms Franks hypothesis about the `flatMap` operator: After the user clicked the reset button, the operator emits multiple values.



[Extension Development Host] - index.ts — experiment

src > problem-1 > index.ts > createEngine

```
      tap((x) => console.log('Before flatMap', x)),
      flatMap(() =>
        merge(
          input.pipe(
            tap(() => setButtonsEnabled(false)),
            tap((x) => console.log('Input', x))
          ),
          counter(increment, decrement).pipe(
            map((count) => `${count}`),
            tap((x) => console.log('counter', x))
          )
        )
```

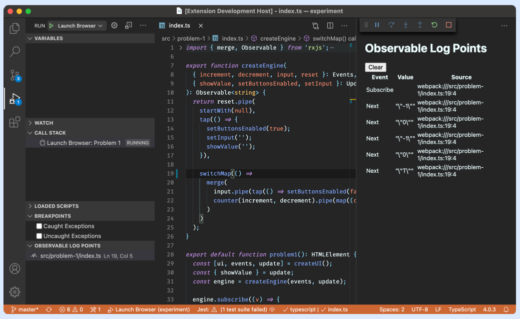
```
After flatMap -1                                              src/problem-1/index.ts:32
counter 0                                                     src/problem-1/index.ts:28
After flatMap 0                                               src/problem-1/index.ts:32
Input T                                                       src/problem-1/index.ts:24
After flatMap T                                               src/problem-1/index.ts:32
> Before flatMap MouseEvent {isTrusted: true, screenX: 324, screenY: 575, clientX: src/problem-1/inde…
302, clientY: 449, …}
counter -1                                                    src/problem-1/index.ts:28
After flatMap -1                                              src/problem-1/index.ts:32
counter -1                                                    src/problem-1/index.ts:28
After flatMap -1                                              src/problem-1/index.ts:32
counter 0                                                     src/problem-1/index.ts:28
After flatMap 0                                               src/problem-1/index.ts:32
counter 0                                                     src/problem-1/index.ts:28
After flatMap 0                                               src/problem-1/index.ts:32
Input T                                                       src/problem-1/index.ts:24
```

Step 5.B **Verify**

Frank launches the application again and repeats the steps necessary to reproduce the reported bug.

The application appears to work correctly now. A quick look at the log point monitor confirms that the previously observed behavior of multiple values emitted is fixed.

Frank has reached his goal successfully.



[Extension Development Host] - index.ts — experiment

src > problem-1 > index.ts > createEngine > switchMap() cal

```
> import { merge, Observable } from 'rxjs';…

export function createEngine(
  { increment, decrement, input, reset }: Events,
  { showValue, setButtonsEnabled, setInput }: Upd
): Observable<string> {
  return reset.pipe(
    startWith(null),
    tap(() => {
      setButtonsEnabled(true);
      setInput('');
      showValue('');
    }),

    switchMap(() =>
      merge(
        input.pipe(tap(() => setButtonsEnabled(fa
        counter(increment, decrement).pipe(map((c
      )
    )
  );
}

export default function problem1(): HTMLElement {
  const [ui, events, update] = createUI();
  const { showValue } = update;
  const engine = createEngine(events, update);

  engine.subscribe((v) => {
```

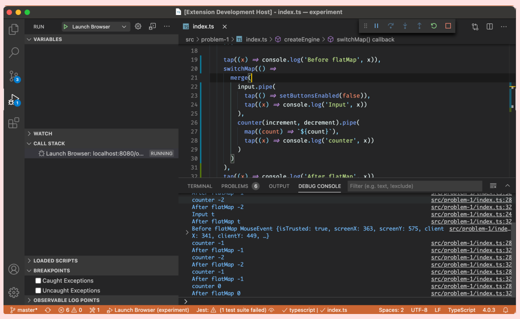
Observable Log Points

Event	Value	Source
Subscribe		webpack:///src/problem-1/index.ts:19:4
Next	"\"-1\""	webpack:///src/problem-1/index.ts:19:4
Next	"\"0\""	webpack:///src/problem-1/index.ts:19:4
Next	"\"-1\""	webpack:///src/problem-1/index.ts:19:4
Next	"\"0\""	webpack:///src/problem-1/index.ts:19:4
Next	"\"T\""	webpack:///src/problem-1/index.ts:19:4

Step 6.A **Resolve Bug**

After researching the RxJS documentation, Frank knows that the `flatMap` operator [3] does not unsubscribe observables it created earlier. This sounds like a reasonable explanation for the observed behavior.

Frank replaces the faulty operator with the `switchMap` [4] operator.



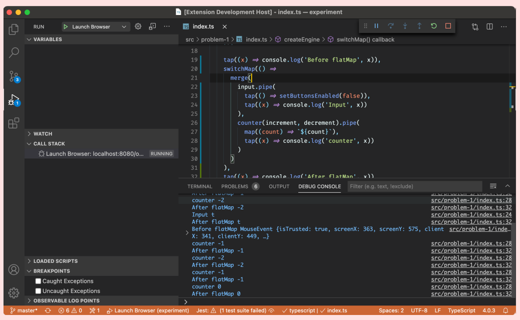
```
    tap((x) => console.log('Before flatMap', x)),
    switchMap(() =>
      merge(
        input.pipe(
          tap(() => setButtonsEnabled(false)),
          tap((x) => console.log('Input', x))
        ),
        counter(increment, decrement).pipe(
          map((count) => `${count}`),
          tap((x) => console.log('counter', x))
        )
      )
    ),
    tap((x) => console.log('After flatMap', x))
```

```
counter -2                                                              src/problem-1/index.ts:28
After flatMap -2                                                        src/problem-1/index.ts:32
Input t                                                                 src/problem-1/index.ts:24
After flatMap t                                                         src/problem-1/index.ts:32
Before flatMap MouseEvent {isTrusted: true, screenX: 363, screenY: 575, client  src/problem-1/inde…
X: 341, clientY: 449, …}
counter -1                                                              src/problem-1/index.ts:28
After flatMap -1                                                        src/problem-1/index.ts:32
counter -2                                                              src/problem-1/index.ts:28
After flatMap -2                                                        src/problem-1/index.ts:32
counter -1                                                              src/problem-1/index.ts:28
After flatMap -1                                                        src/problem-1/index.ts:32
counter 0                                                               src/problem-1/index.ts:28
After flatMap 0                                                         src/problem-1/index.ts:32
```

Step 7.A **Verify**

Frank launches the application again and repeats the steps necessary to reproduce the reported bug.

The application appears to work correctly now. A look at the trace log confirms that the previously observed behavior of multiple values emitted is gone.



```
    tap((x) => console.log('Before flatMap', x)),
    switchMap(() =>
      merge(
        input.pipe(
          tap(() => setButtonsEnabled(false)),
          tap((x) => console.log('Input', x))
        ),
        counter(increment, decrement).pipe(
          map((count) => `${count}`),
          tap((x) => console.log('counter', x))
        )
      )
    ),
    tap((x) => console.log('After flatMap', x))
```

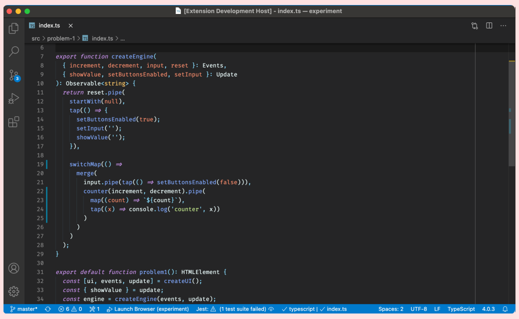
```
counter -2                                                              src/problem-1/index.ts:28
After flatMap -2                                                        src/problem-1/index.ts:32
Input t                                                                 src/problem-1/index.ts:24
After flatMap t                                                         src/problem-1/index.ts:32
Before flatMap MouseEvent {isTrusted: true, screenX: 363, screenY: 575, client  src/problem-1/inde…
X: 341, clientY: 449, …}
counter -1                                                              src/problem-1/index.ts:28
After flatMap -1                                                        src/problem-1/index.ts:32
counter -2                                                              src/problem-1/index.ts:28
After flatMap -2                                                        src/problem-1/index.ts:32
counter -1                                                              src/problem-1/index.ts:28
After flatMap -1                                                        src/problem-1/index.ts:32
counter 0                                                               src/problem-1/index.ts:28
After flatMap 0                                                         src/problem-1/index.ts:32
```

Step 8.A **Revert Hypothesis Instrumentation**

Frank removes all `tap` operators he added solely for the trace log generation, except the one in Line 24.

Another engineer notices this leftover during the code review, fortunately. Frank removes the forgotten statement afterwards.

Finally, Frank has reached his goal.


[Extension Development Host] - index.ts — experiment
index.ts
src > problem-1 > index.ts > ...

```typescript
 6
 7  export function createEngine(
 8    { increment, decrement, input, reset }: Events,
 9    { showValue, setButtonsEnabled, setInput }: Update
10  ): Observable<string> {
11    return reset.pipe(
12      startWith(null),
13      tap(() => {
14        setButtonsEnabled(true);
15        setInput('');
16        showValue('');
17      }),
18
19      switchMap(() =>
20        merge(
21          input.pipe(tap(() => setButtonsEnabled(false))),
22          counter(increment, decrement).pipe(
23            map((count) => `${count}`),
24            tap((x) => console.log('counter', x))
25          )
26        )
27      )
28    );
29  }
30
31  export default function problem1(): HTMLElement {
32    const [ui, events, update] = createUI();
33    const { showValue } = update;
34    const engine = createEngine(events, update);
```

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, New York, NY, USA, 15–24. DOI: https://doi.org/10.1145/3427763.3428313
2. L. Layman, M. Diep, M. Nagappan, J. Singer, R. Deline and G. Venolia, "Debugging Revisited: Toward Understanding the Debugging Needs of Contemporary Software Developers," 2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement, Baltimore, MD, 2013, pp. 383-392, doi: 10.1109/ESEM.2013.43 .
3. ReactiveX. 2020. RxJS - flatMap. Retrieved 05-December-2020 from https://rxjs.dev/api/operators/flatMap. Version cbc77213e97ecc00d90a65ecf18707b76ebfe7fc.
4. ReactiveX. 2020. RxJS - switchMap. Retrieved 05-December-2020 from https://rxjs.dev/api/operators/switchMap. Version cbc77213e97ecc00d90a65ecf18707b76ebfe7fc.