I Fixed A Bug The Other Day

I fixed a bug that was in production for 2 years the other day. My boss said thank you, the UX researcher that called it out two years ago and was told it was not possible to do on our end said thank you. The product manager that had tried to tackle the issue a few times but gotten nowhere with their devs said thank you. It turns out that it was one of the biggest customer complaints and for some reason it had been neglected? ignored? idk. The change was about 20 lines of code, as they usually are, but had an outsized impact on customer satisfaction ratings. Why?

The Issue

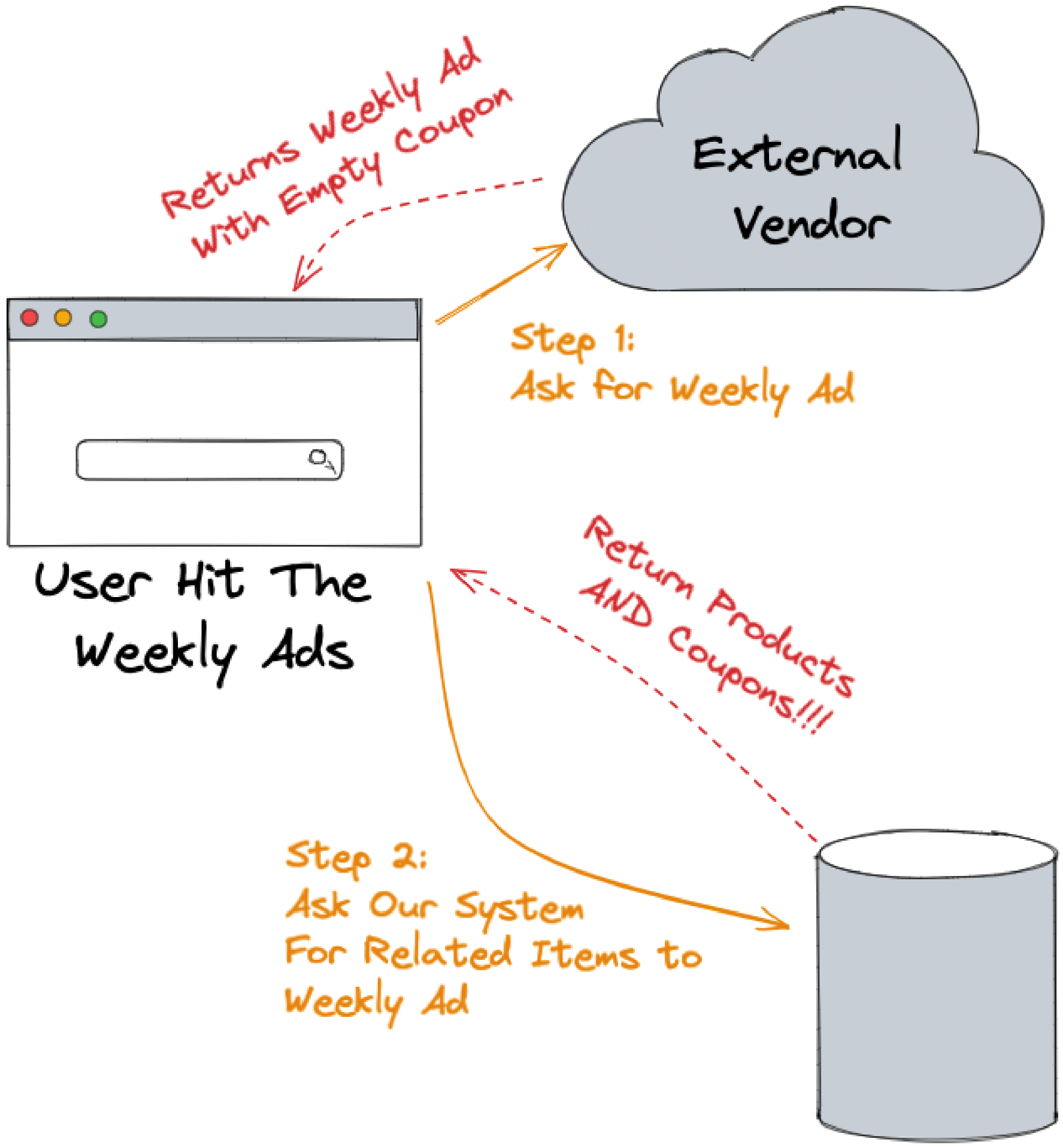
I work in supermarket systems. Think e-commerce on steroids. We do weekly ads (which are called circulars) and these have discounts, the app also handles enough work to employ hundreds of developers working on loyalty programs, personalized recommendations, organizing pick up, delivery, managing availability for orders that are fulfilled in a physical location (which is most orders) shopping lists, a custom cms..., you get the picture. The issue was that our system for weekly ad was having an issue where the ads needed to 'clip' a coupon (read hit an api) for the discounted price to reach our cart calculations. However, the api that we used for managing the weekly ads was managed by an external vendor, and often when we queried their endpoint, the coupon field was returning empty when it shouldn't. To the customer, this meant that they would click on a tile that says a discounted price in their description, but would hit their cart as full price. Really frustrating and a cause of many loud customer complaints.

Why was it hard to fix

It wasn't. It was, however an issue that was difficult to replicate in our lower test environments, as the data simply wasn't there. It was also an issue that had to do how the external provider

formatted the data that they were passing to us. Because this data was missing a field, it required manual data entry to update it. However, this data changed in real time, so it was OFTEN out of date. Whenever this issue came up to developers, the reaction was always to blame the vendor. This is natural, as it technically was the vendor's fault. However, what became apparent is that people were not going below the surface level to get to the heart of what could be done.

It became clear that this was an issue in **process** not in **execution**. The inertia in the company was to find why it would be hard to do, or impossible to do, why the fault lay elsewhere. We had spikes, and backlogs with this issue, but no action. I had never looked at this code, but was told to take a look, given that we seemed to have the data coming back in one of our API calls. Here's a diagram of the flow of info that we already had on our site:



That's right, for any developer, the answer here is simple: use the coupons that we are ALREADY receiving on the frontend to populate the fields that users expect so they can get their discounts. Products already had the associated coupons, but we were simply ignoring that. Not only where we ignoring that, we said it was not possible to do. This reminds me of this article I read on how to report a bug:

> Deny that there is a problem.
>
> Deny that it is your problem.

Ask for more information.

Complain.

The real difficulty was in reproducing the data, so I completely hacked it, and replaced the api call service with a dumb function that returned hardcoded data copied from a production. As our principal engineer said to me when I told him about the hack to get the data in my local environment: "**All data in lower environments is mock data**".

The Solution

There is very little that is exciting about the bugfix. It was easy, a few array methods to ensure we were not searching for duplicates, and it even required one less API call than before. It worked nicely and the customer satisfaction scores went up as expected. What I did is come in as an outsider to the issue and this particular business concern, let my naivety guide my investigation and provided a solution within a few days. Why wasn't this done before is the more interesting question, and it seems to be related to a few things that are above my paygrade but are quite interesting to me.

Closing Thoughts

I like my company, and my direct managers, and several other people that I interact with in a regular basis. However, this was a case where the inertia of the status quo got the better of the organization at the expense of customer experience. I want to understand how this came to be, rather than to shift blame around, and to do there are a few things that I am paying attention to. The incentives and the feedback loops.

A company that is not a startup has strange incentives opposed to some of the original ones when it is getting started. Here's the cycle described by Balaji Srinivasan:

> First, a libertarian(ish) founder leaves the stifling bureaucracy of a big company to start their own. Most immediately fail, but through pure maneuver warfare and relentless execution, that

> founder might be able to make enough money to hire someone. In the early days the most important quantity is the burn rate. Every single person must be indispensable.
>
> Eventually, if successful, the company starts building up some structure. Conservativism takes over. With the business growing consistently, the founder adds structure, career tracks, and a stable hierarchy. Now the most important quantity becomes the bus number, the number of people who can get hit by a bus such that the company is still functional. Suddenly every single person must now be **dispensable**.

The bus factor creates redundancy, and people being dispensable makes a strange feedback loop. When everyone is dispensable, as a good company should make their employees, there is a tension between owning your work and standing out, vs just doing the minimum. Where that line sits is not clear at all.

This pressure and quite frankly this contradiction is an extremely difficult one to solve. Harvard and others have written about it Why Big Companies Can't Innovate. I am interested in this, because I am interested in doing work, at scale, within a company that has a culture of solving problems, not of being a behemoth with enough inertia to not need to correct when there is a need. Luckily, my team within the larger org values this nimble attitude of getting sh(stuff)t done, and was able to in some sense circumvent this inertia and this extreme risk aversion.

I've also been thinking quite a bit of a company as a cybernetic system. You have a product, and that product has a shape which is shaped by the organizational structure (in Software Development, we call it Conway's Law). The product reflects the org, and the org reflects the product. This separation gets reinforced, and we were reinforcing invisible (and nonexistent to the customer) boundaries within ourselves and the vendors for no good reason. I want to investigate this notion of cybernetic systems and how to direct them in the context of a software company, so I'll be writing more about that.

In the end, this issue revealed a flaw in process more than a flaw in the technical execution of our product. Of course a leaky abstraction showed to the customer, but it was a matter of understanding it and patching it rather than allowing that distinction to continue and reinforce itself. As I progress in software, I am very interested in thinking of steering the cybernetic system

of a corporation in order to get self reinforcing outcomes in an intentional direction. I will be writing on that more soon.