



Oh God I'm Not Getting a Return Offer

Why You Should Really Practice
Requirements Engineering



Who Am I?

- Henry Pabst - Graduate Student
- Graduated with BSc. in Computer Science from University of Alberta.
- Dan said I can use a fun or casual tone.
 - He may regret this after the presentation.



UAlberta Vs. UWaterloo

- UAlberta has a co-op program, but it sucks.
- One internship before graduating, helping with research in Department of Construction Engineering.
- Not a CS school, no fancy tools like UWFlow to check out classes ahead of time.




Speaking of UWFlow...

- “Worst course I've taken in UW. The knowledge is more useless in the practical software world than EARTH 121. A course on writing effective emails would be more useful than this course.”
- “The content is frankly useless”



<https://uwflow.com/course/se463>

<https://uwflow.com/course/cs445>



What Does the Director of Software Engineering Have to Say?

- “...this course comes up extremely often as a pain point; it is the most-often cited ‘don’t like’ in the 2017 exit surveys. UWFlow ratings are abysmal.”
- “I submit that [Software Requirements] doesn’t need Mother Theresa. Kenny Wong at Alberta is definitely not Mother Theresa and offers a 4-week Coursera course with Coursera ratings of 4.7”



Who Here is Not Required to Take This Class?

- I know of myself and one other grad student.
- Any others?

The UWaterloo Experience - An Outside Perspective





Is This Presentation Going to Be Entirely Complaining About Us?

- No, there's a point to this.
- UWaterloo students come into this class from a perspective of experience and privilege.
- But Software Requirements is an important part of software development.
- So, I'm going to give you a retrospective on what can go terribly, terribly wrong.



Previous Experience

- Internship with Department of Construction Engineering.
- Completely independent, no formal requirements engineering processes.
- C# and VB.NET basic simulations and Client-DB applications.
 - Why this old technology? It was all my manager knew.
- I received an outstanding performance review!



Summer Internship

- Summer 2019
- Online retailer (had to sign an NDA).
 - Let's call it Really Big Retailer, or RBR.
- Backend development in Payments department.



First Day, What's My Project?

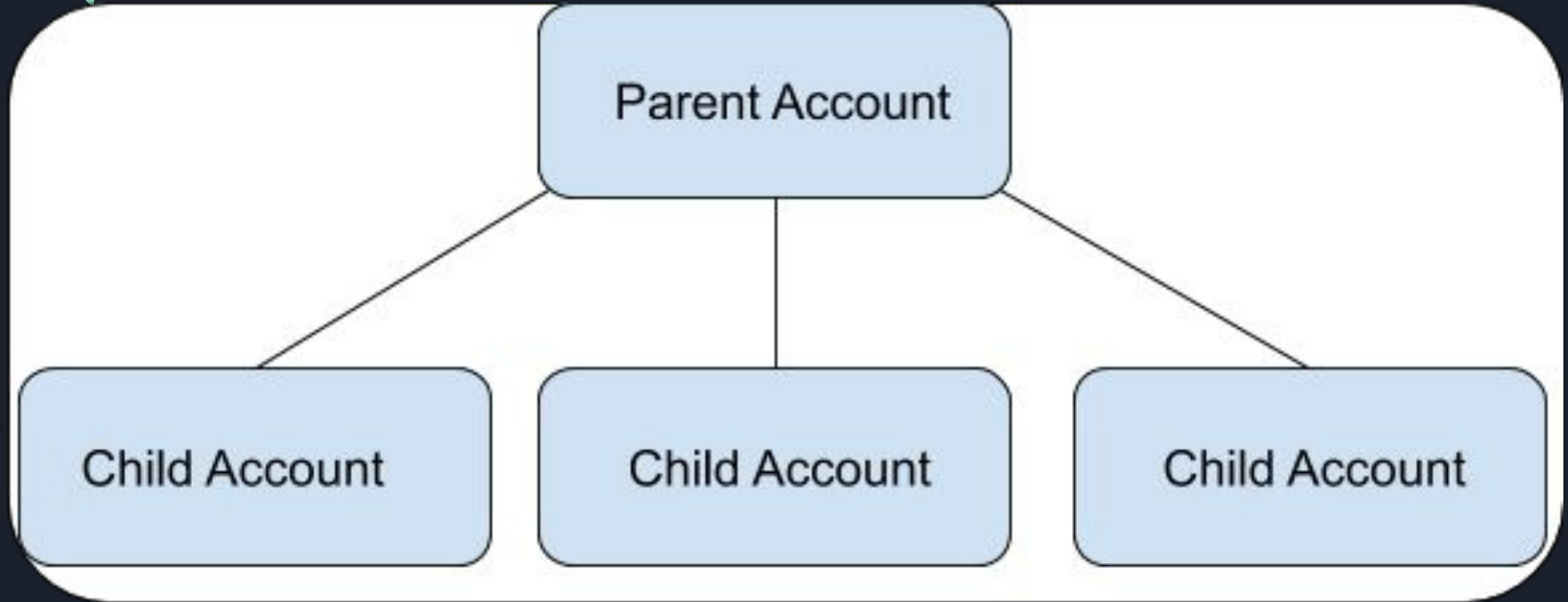
- RBR doesn't do payment processing itself, works with banks and other companies to accomplish this.
- Operations team has a problem, too many API keys to manage for a payment processor.
 - ~80 API keys rotated monthly.
 - Lengthy manual process for each rotation.
 - Integrate a new API that reduces the number of keys to manage.
- I handle the entire project from requirements to deployment.



First Day, What's My Project?

- Payment processor's new API lets you link accounts in a many to one parent-child hierarchy.
- Can perform transactions for the child account with only the parent's API keys.
- Account linking has to be done manually.

First Day, What's My Project?



A golden retriever is sitting at a desk in an office setting. The dog is wearing a pink and purple patterned collar. It is looking towards the left of the frame with a slightly open mouth, appearing confused or uncertain. In front of the dog is a laptop with a blue screen and a keyboard. Behind the laptop is a larger monitor. The background shows a desk with a printer and a lamp.

I HAVE NO IDEA

WHAT I'M DOING

How Does RBR Operate?

- Each team decides their own processes and style.
- “There are no rules, only guidelines.”
- Most teams used some variation of Agile.
- My team also used Agile and had no procedures for requirements engineering.





How Does RBR Operate?

- Typically, the manager would handle gathering requirements and provide them to the team as user stories.
- But not for the lowly intern!
- Everything is my sole responsibility.



First Step: Requirements Elicitation

- Started with artifact-based elicitation.
 - Good idea as a first step. I needed to understand the existing RBR systems before I could understand what was and wasn't possible on the project.
 - Existing artifacts comprised a company-wide wiki and internal StackOverflow.



First Step: Requirements Elicitation

- Identified stakeholders.
 - Not just the operations team that had requested the project, but my development team that was responsible for maintaining the project once my internship was over.
- Set up an interview meeting with representatives of the development and operations team.
 - I had the project request documentation, but I still wanted to nail down the full functional and non-functional requirements.



First Step: Requirements Elicitation

- Already I'd made a pretty major mistake that would cause issues later in development.
- What is it?



Requirements Elicitation Problems

- Only interviewed a single representative from each stakeholder group.
- I focused on leading/dominating the discussion as I was the lead on the project.
- I assumed that the requirements given to me were correct and largely complete.
- I created no artifacts from this meeting aside from some notes in a paper notebook.
 - No SRS, no user manual, nothing to point to later and say “these are the requirements”.



Requirements Elicitation Problems

- Most of the requirements gathered were quality-based non-functional requirements.
 - “Make the website easy to use.”
 - “Make sure the changes you need to make don’t slow down our existing systems.”
- Despite this lack of clear functional requirements, vague non-functional requirements, and non-existent documentation, I moved into the technical design phase.

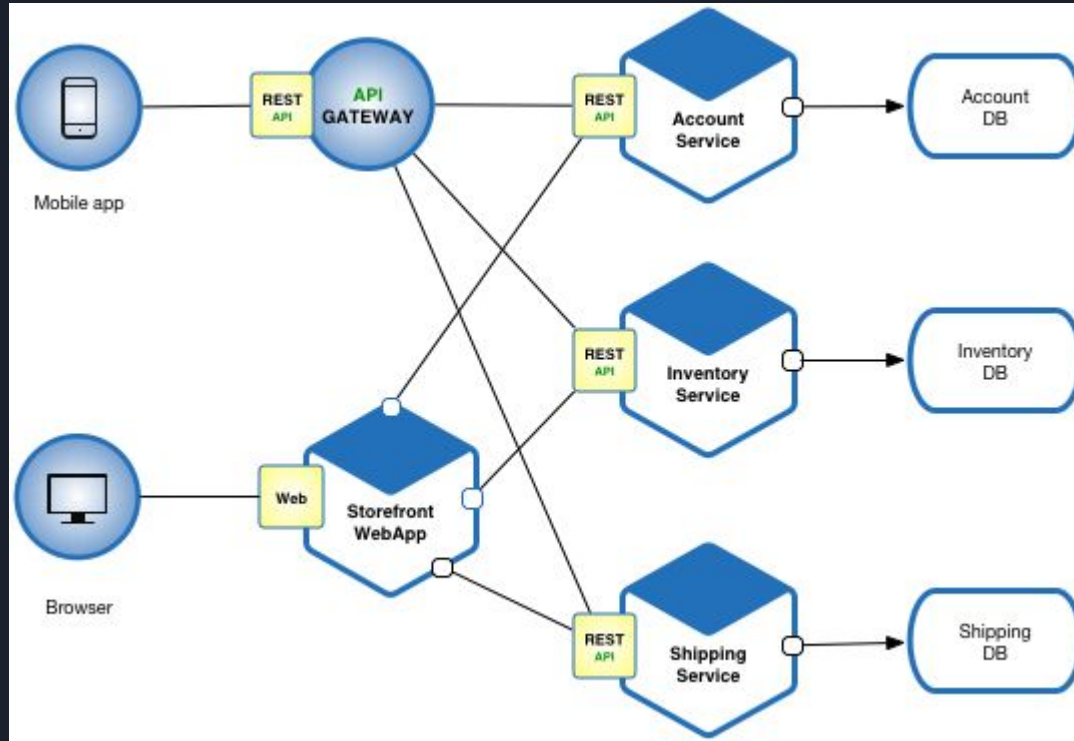


Technical Design

Two major sections of the project:

1. Back-end changes to existing systems.
2. New internal website that would allow operations team to link accounts.

Back-End Technical Design



Back-End Technical Design





Upstream and Downstream Services

- My team controlled the final service before we interacted with the payment processor's API.
- I could fairly easily integrate the new API at this point, but we needed additional data from our service's users to make calls to the new API.
- Simple design: make our users provide the data. Modify the interface to require the data in requests to our service.



Upstream and Downstream Services

6/7ths of the team's response:

“I don't care, that sounds fine.”

The 7th team member's response:

“Oh God no. Why did we take on an intern?”



Changing Requirements

- I'd planned, and made, changes to other team's services with the assumption that I would be able to modify the interface of the final service.
- I didn't work with the entire development team, as such I missed out on the requirement "Don't modify our interface" from one of the main stakeholders.
- Several weeks of work, including 11pm meetings, wasted.



Conflicting Requirements

- I not only failed to elicit near-complete requirements, I elicited conflicting requirements.
- Operations team wanted a fully-featured “one-stop-shop” type website to make account connections, view account analytics, etc.
- Development team wanted the absolute bare minimum implemented since they would have to maintain the website in the future.



Conflicting Requirements

- Operations team wanted the website to be written up in the latest technology, have fancy animations and dashboards, etc.
- Development team wanted a static website written in what they already knew, Java. (Java web development sucks).



Conflicting Requirements

- For once, I got lucky.
- I went forward with a website with a React front-end and NodeJS back-end with the bare minimum number of features.
 - I was way more familiar with NodeJS and React than I was with Java web development.
- Eventually, the manager leaned on the operations team strongly enough that they gave up on having all the features they wanted.
- What should I have done differently?



Cost Estimation

- After the technical design phase, I had to come up with cost estimations for each part of my project.
- Like every other process at RBR, cost estimation was done entirely informally.



Cost Estimation

- Cost estimation within my team was done through a quorum process.
- The team would take user stories, and assign them “points” based on how long each member thought the user story would take.
- A vote would be taken on how many points each team member thought a story should have and the majority won.



Cost Estimation

- For me, no such cost estimation process.
- Hadn't heard of COCOMO, function point analysis, how to analyze effort required from previous projects, etc.
- Just ended up going by “gut instinct” on how long I thought each story would take.
- Which of the cost estimation methods that we used probably would have worked best?



Cost Estimation

- Trick question!
- If we wanted to stick with the points system, there's not much we can do.
 - Points don't directly translate to lines of code, only to estimated time required for a given task.
 - User stories can be tackled asynchronously: one story can be tackled while waiting on a code review or a meeting for another.



Cost Estimation

- Best chance would probably be to try and work with other team members to estimate points based on their experience working on previous projects.
 - But this doesn't take into account the fact that I'm working on unfamiliar systems, using unfamiliar technologies, etc.



How Did “Gut Instinct” Work Out?

- Not very well.
- The original timeline was a 4 month internship: 1 month to learn the existing systems and gather requirements, 2 months to do the actual project implementation, 1 month to do a small follow-up project.

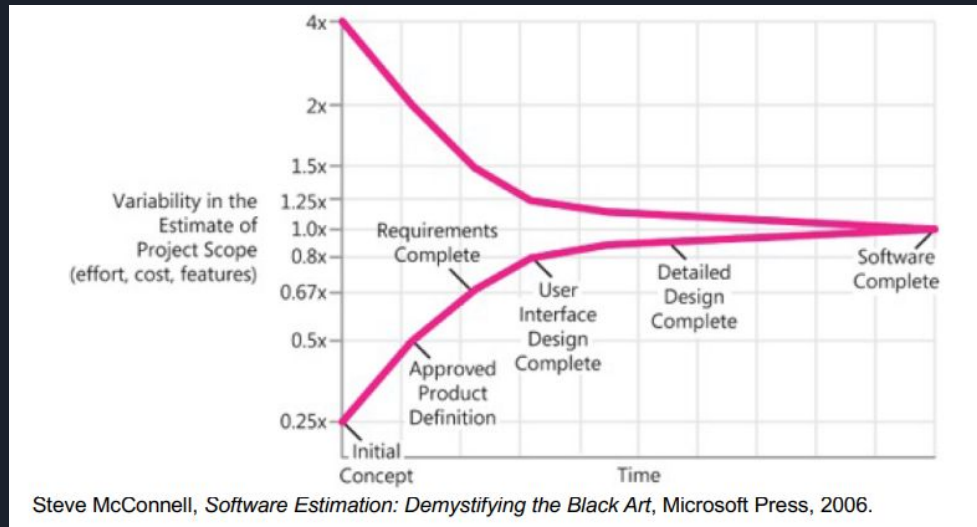


How Did “Gut Instinct” Work Out?

- It ended up being 3 weeks to learn systems and gather requirements with the rest of the internship spent on the original project.
- Project still wasn't done when the internship was over.

What Went Wrong With Cost Estimation?

- I made my gut instinct estimations at the start of the project and they turned out to be wildly off.





What Went Wrong With Cost Estimation?

- I wildly underestimated the time for a required security review.
- Since I was handling API keys for a payment processor, the website I developed had to go through the strictest security review RBR could manage.
- The time allotted for the review was 2 weeks, it ended up taking longer than 1 month.
- Review was still ongoing when my internship ended.



Summary

- “You kinda screwed your project over.”
- Absolutely! What was originally estimated to take 2 months for implementation ended up taking 4 months before completion.
- In retrospect, most of the blame can be placed on poor requirements engineering.
- A user manual or SRS for the project would be great for my team members, but no documentation of the requirements process exists!



Summary

- Despite all these mistakes, I ended up getting a return offer.
 - Bribery was involved.
- Questions? Comments? Concerns? Mockery?