**DevOps thoughts**

**Role of operations**

Operations should be much more involved in the product. Operations at RBI still has a service mentality. If I wanted them to do anything, I raised a ticket and it would be assigned to an operator. So if there was a production issue for example, I raised a ticket and it would be assigned to a DBA. The DBA would not necessarily know anything about the product and would end up having to trouble-shoot it myself and all the DBA was doing was giving me access to the server. Which wasted our time. Generally, I did have access to the same DBA but that was more by accident.

What I would like is a DBA to engaged in product development. They would contribute to the technical design. They could pick up a production issue immediately and, because they know the product, could identify the issue and feed-back to me as soon as possible if there was a development issue. Also could do other stuff like check what indexes were being used and where an index may be required.

Also, they weren’t implementing infrastructure as code (not for SQL anyway), each environment was manually created by them or us and there was no guarantee they would be the same (in fact they weren’t the same which always caused issues.)

**Example of where DevOps could help**

Most production issues were caused by SQL “hanging”. A SQL statement that runs every day in minutes will suddenly not run at all (this in itself was frustrating, SQL doesn’t fail gracefully.) If you ran an estimated execution plan on it, it would almost always reveal that the SQL needs an index. If I was to do this properly, I would have had to:

1. Identify the SQL that was “hanging” (reasonably easy to do as everything was logged.)
2. Raise a ticket for a DBA to get an execution plan on this SQL (this could take a couple of hours.)
3. At the same time, raise a ticket to get a backup of live (this would take a couple of hours.)
4. Ask the developer to restore the backup of live
5. Run an estimated execution plan on the SQL in the backup. Normally this didn’t reveal that an index was required (because the environment was not exactly the same as Live.)
6. Once we got the results of the estimated execution plan, create the index in dev and run that part of the job to ensure it had no adverse effect.
7. Create an emergency release which involved sign-off from one senior management in operations and my manager
8. Release to OAT
9. Release to Live
10. Test in Live to ensure this fixed the issue.
11. If this took more than one day then we would have to restore missing data.

I got around this by giving myself the rights to execute SQL plans on live and to create indexes. Which I shouldn’t have had. I was working against the system because the system wasn’t working for me.

How would DevOps help?

* DBA engagement - DBA would be more involved in product design and would be able to advise (maybe) when this sort of situation may arise.
* Containerised environments - When a production issue arose, I could spin up an exact replica of Live within an hour (time it takes to backup and then restore live databases plus create the environment.) I could trigger this myself.
* Continuous delivery pipeline - Developer could then quickly dev a fix for it (using hotfix workflow) and we could deploy through the environments and on to Live ourselves. No emergency release required. No sign-off from management.

**Use of Telemetry and advanced statistic techniques to identify problems**

We had a problem in understanding when a feed of data was not supplying complete data. For example, the amount of form data fluctuates every day. How do you know if there is missing data say a particular form is not supplying data? We relied on visually examining the numbers everyday which was far from satisfactory.

We could use advance statistic techniques to be able to identify when where was a problem for example by analysing natural peak and trough of form submits and using that as a basis for monitoring. These skills were available and were probably already being done by data analysts in the markets. The spirit of dev ops means we could use these skills/techniques on our production environment to identify issues. DevOps is about working in partnership with other teams to leverage skills and know-how to meet a common goal.

**Developing a high-trust culture**

DevOps doesn’t get far unless there is a fundamental change in culture away from command-and-control. Releases requiring authorisation slow down releases. In my experience, release authorisation was meaningless, no one authorising the release had any knowledge of the release and weren’t using anything to actually authorise it (literally a tickbox exercise.) Changes should be peer reviewed because these are the individuals who have an understanding of what was being released.

**What I learnt from More2**

More2 couldn’t afford to have multiple environments so generally we had Dev, Test and Production. There was no split between OAT and Test. So you didn’t really know how something would perform when you released it to test.

The ETL tool didn’t work very well on Dev and didn’t work at all on Test. It was no one’s job to ensure that it worked. It was down to individual dev to attempt to sort out issues as they came about. More2 could not afford to have developers to maintain the ETL tool. Developers had to be working on what made More2 money. However, the fact that the ETL tool didn’t work on Test meant introducing a risk. And developers would spend a lot of time on getting the tool to work on Dev which meant development was often delayed.

Releases often failed. For a number of reasons

* No OAT environment that matched Live .
* No analysis of requirements of creation of a technical spec. This meant something would be released and then we’d find out it didn’t quite match requirements.
* No regression testing
* No functional testing apart from that done by the developer themselves
* Couldn’t run the whole “build” before deploying to live. Build would generally fail on other issues in Dev environment and fixing these would delay releasing work. Also, build would take too long on Dev environments.
* Not clear before hand who had to sign off a release and what they required

No Architiecture. Head of Development was de-facto architect. We had session describing how to implement a solution with data flow diagrams. But it was no one’s job to record and document this (HoD certainly wasn’t going to do this.) So, 2 months later, we’d have the same discussion again because no one can remember what was decided (at best someone would take a photo of it)

No overriding architecture. Each client had their own version of the build database. No real concept of an ETL staging, and DW stage. It was incredibly difficult to reverse-engineer a build database. Really we were reliant on knowledge held by key developers.