Rethinking DevOps for legacy systems

People often talk of apps as something new – an area of IT development that has only been around since smartphones became commonly used, but that's not right.

It's true that mainframes often look like legacy systems. Mainframe are computers used primarily by large organizations for critical applications; bulk data processing, such as census, industry and consumer statistics, enterprise resource planning; and transaction processing. They are larger and have more processing power than some other classes of computers: minicomputers, servers, workstations, and personal computers. The infrastructure is quite complicated and the qualifications to work on these systems are quite specialized. It's not an easy environment.

Some of the key problems developing in the mainframe environment are:

1. DevOps pain; a lot of manual operations for code building, customization, and setup for various environments.
2. Development cycle; typically the cycle is between a week to a month – it's not a rapid development environment.
3. Version control; we have systems to help, but nothing is integrated with the modern version control systems most development environments use today.
4. Limited automation.
5. Poor visibility and control at all stages of development.

If you also work with mainframe development then you might know about these problems already. So what did we do in our own development environment to try addressing these problems?

1. Automation; we started using the UrbanCode family of tools to start automating some of the infrastructure tasks.
2. Integration; we integrated the UrbanCode processes with the Rational tools family – RTC Rational Concert and RQM- Rational Quality Management.
3. Reducing tools; we reduced the number of tools being used so we could focus on using the remaining ones more effectively.
4. Scalable Pipeline; we built one project using the new DevOps methods and then assisted all teams to develop their projects this way, so these methods scaled across all development teams.
5. Security; increased automation left gaps in security so we used DevSecOps to embed security functionality.

I can talk in detail about how we did all this, and the benefits we found, but for the sake of this blog it's better to just highlight the main benefits we found from this approach to DevOps.

1. Faster deployment; it's faster to develop new processes and systems therefore your business operations can be more efficient.
2. System Thinking; building this DevOps environment creates a culture of system thinking which means that responsibility, transparency, and feedback are all improved. Systems Thinking creates a much more focused team that works together.
3. Increased Effectiveness; IT development is typically full of waste. People are waiting on others to deliver and they cannot work until a specific part of a project is handed over.

Managing pipelines makes deliveries more predictable and allows resource to be allocated more effectively.

4. Better Quality; We now have more tests, more automation, and User Acceptance Testing. We also trust the pipeline. People are used more effectively and this also increases the quality of deliveries.

We know from our own experience that our team now spends 20% less time on unplanned work and reworking problems. This has led to a 40x reduction in systems failure and the team is 50x more satisfied with their work. Even when a failure occurs, we can now recover 20x faster than before.