How to Improve Your Software Release Management Process – A Real-time Case Study

Software projects take investment, support, development and a lot of hard work and dedication. Good release management practices ensure that when your software is built, it will be successfully deployed to who want to use it. You have the opportunity to retain existing customers and hopefully to win new clients.

One of my major clients had a problem. It needed to implement business critical releases on time, which required it to reengineer its billing and account management systems.

These systems had to be in place within three months, this is highly business critical to the client and company as well. But the release management processes were poor, and it was extremely problematic and inconsistent.

The company hired me to help deliver the release process within the time constraints and to turnaround a failing release management process. Within four months, we'd released both the pending releases and two scheduled releases of the reengineered applications.

And our team established a clear-cut and lightweight release management process to ensure that future releases would happen on time and to the required quality. We followed these below steps to overcome from the critical situation to systematic process implementation stage within 6 months.

Step-1 Understand the existing shape of release management.
We can't begin to repair something without understanding what it is, and how and where it is broken. The first step in improving our Business Unit's release management system was to form a detailed picture of the current release process with a number of walk-through sessions and gap analysis with key individuals involved in their respective technologies and towers.

From this gap analysis we determined that our starting point was pretty bad. When we joined the project, there was no proper documentation available, software builds still waiting to be deployed few weeks after being completed, Test environments were limited and not managed, so they were regularly out of date and could not be used. Worse still, it took a relatively long time to turn around new environments and to refresh existing ones.

When we arrived on the scene, regression testing was taking up to two months to manually execute. It was usually dropped, unit testing was performing developer's individual machines, there was no such version control mechanism and folder structure for before unit testing and after unit testing, this kind of process significantly reduces the quality of any software that made it to release.

Overall, morale and commitment were very low. These people had never been helped to deliver great software regularly, and it had worn them down.

Step-2 Initiated a regular release process cycle.
Once we got a picture of the current state of the process, we set about establishing a regular release process it is vital because. *If the development team is the heart of the project, the release or configuration life cycle is its heartbeat.*

In determining how often to release into production, we had to understand how much nonfunctional testing was needed and how long it would take. This project required regression, performance and integration testing as well, It creates an opportunity to meaningfully discuss nonfunctional testing that the software may need. It announces a timetable for when stakeholders can expect to get some functionality. If they know that functionality will be regularly released, they can get on with agreeing what that functionality would be.

It creates a routine with which all teams can align (including marketing and development) and It boosts customer’s confidence and trust that they can order something and it will be delivered on time. Your release cycle must be as accurate as you can make it, not some pie-in-the-sky number that you made up during lunch. Before you announce it, test it out. *There is nothing worse for a failing release process than more unrealistic dates!*

Firstly we started out by suggesting a weekly release cycle. That plan proved unfeasible because of few technical issues at client’s end, the client's database environment could not be refreshed quickly enough. Secondly we tried two-week release cycles.

There were no immediate resistance from the onshore and offshore teams, but it failed the first two times! In the end, two weeks was an achievable cycle, once we overcame some environment turnaround bottlenecks and automated some of the tests. *Here is what I mostly applied few release management techniques which I adopted during my past experiences.*

Finally we established a cycle whereby, every two weeks, production-ready code from the development team was put into system test. Then two weeks later, we released that code into deployment and production environments.

Here the development teams have learned a lesson that “your release cycle is not about when your customer wants the release. It's about when you can deliver it to the desired level of quality”. Our customers supported our release cycle because we engaged them in determining the cycle. There is only one consideration in determining the release regularity.

Step-3 Initiated lightweight processes in place.
Review your results regularly and then do some more. Repeat this cyclic approach until you get the results you want. This is what we called a lightweight release process.

Lightweight processes are those that do not require lengthy documents, approvals or endless meetings to get agreement. They usually require only the minimum acceptable level of inputs and outputs. *Underpinning this approach is the tricky issue of documentation. You need to record what you did and how you did it. Otherwise, what do you review and how do you improve?*

I don't mean the kind of documentation that puts its readers to sleep. I mean documentation that people can read and act on simply without any dependency from other geographically distributed teams. The release management team uses their experience to create minimal but effective documentation of what onsite or offshore leads or delivery managers were agreeing to build in every cycle of work. They record what they built, how they built it and what was required to make it work, we saw the value in this approach and rolled it out to everyone else involved in the process.

Initially, we suggested a sequence of tasks to release the software we got from the development teams. It covered how we took delivery from the source control management tools, what packages would be called and how each element (executable code, database scripts, etc.) would be run and on which platforms. Then we did a mock run, using dummy code for each element. We tested our sequence, documenting what we did as we did it, this formed the basis of the installation instructions.

The next step was to get the people who would be deploying the real release to walk through another mock run, using only our documentation. They extended, amended and improved our instructions as they went through. The process became a more inclusive one where everyone contributed to the documentation; since they'd been part of its definition, the process became more widely adopted with better quality.

After each release, we reviewed the process. We examined the documentation and identified changes made during the release. Every time, we looked at how the documentation could be improved and fed the enhancements back into the process. This is how we tested them early and reviewed them regularly.

Step-4 Established a release infrastructure early.
Your release infrastructure is anything that needs to be in place to deploy the software and to enable users to use it, *Your obligation to the customer is not just that you build great software, it is that it's timely available for them to access and use.*

Crucial to getting a good release process is figuring out what you need to have in place to make it available to the customer, *before* the development team is done building the software, the release team ensures that the infrastructure covers the hardware, storage, network connections, bandwidth, software licenses, user profiles and access permissions.

Human services and skills are part of the release infrastructure, for example, if you require specialist software installed and configured, it's not smart to exclude the availability or cost of getting such skills into your infrastructure plan.

It is critical that as a release manager you discover, as early as you can, hidden bottlenecks in procuring the required hardware or the missing skills (say, to configure secure networks from the network team). You need to resolve them before they hold up your delivery.

This is not an easy task. We strove to get our release infrastructure in place as soon as we started on the project. Even after six weeks' lead time, we were still waiting on specialist memory and hard drives for the test servers!

Step-5 Automated and normalized as much as we can.
Automation enables you to do repetitive tasks without typing up valuable human resources, normalizing ensures that your automation's inputs and outputs are consistent every time. I found some other serious errors, a new package was not guaranteed to be the same as the last one; in fact, it was not even guaranteed to be the software they had been building, much less guaranteed to work! It often took the tech staff days to create a package with the features they were delivering in a structure that could be deployed.

We immediately drew up a structure and acceptance criteria for the deployable package the team was delivering to the release team and helped them standardize its packaging. This triggered the implementation of automated processes to build the software in that consistent structure for every release point.

Suddenly, the packaging of the software for release was not even an issue. Because we had automated the verification of the acceptance criteria, for example, that code must be unit tested prior to delivery and test deployed to ensure that it could be deployed; we had guaranteed its executability. As a result, we were able to package, version, and test and deploy finished code with a single command in a very short time.

But automation did not stop there. With each development cycle, we had even more regression tests to do. The existing regression tests would have taken three months to manually execute; as a result, the releases were never properly tested.

Our newly established release cycle found this gap and meant that a release had to be regression, performance and integration tested in two weeks for us to be able to release it into production. We could overcome the different types of testing (integration and performance) by having separate environments for each type. But how would we accommodate three months of regression tests into a two-week window?! Here exactly I applied my most favorite technique which I applied many times.

First, we initiated a prioritization exercise. The customer identified the highest-priority regression tests: the minimum they would accept as proof that the old functionality still worked. Then we set about automating this set. Subsequent acceptance tests also became automated, ensuring that we could regression test every release in hours rather than days.

Step-6 Established positive expectations.
If getting software released is important to you, don't keep it a secret. Release management teams improved their commitment to deliver the software release when they knew it was important.

We backed up this importance by establishing that the designated release manager would expect the software to be ready when the teams agreed it would be ready. We got the program manager to explain to the teams why the release was important.

We requested that the software delivered by the development teams conform to a standard (versioned, tested, documented and packaged); we established that we would request this standard package for every release cycle. We needed to explain why we wanted the software in this way (it made our automated process easier and more consistent) and we integrated the team's feedback into the process.

Establishing positive expectation is a really good way to empower everyone involved in the process. We were not given any executive authority, so there was no fear of sanction or sacking. Instead, we tapped into the power of positive expectation to get people on board to help us improve the release process. We had individuals making key decisions which they never felt able to make before.

Conclusion
No matter how much you spend on hardware, software and fancy processes, without the commitment of team members you will not enjoy sustainable success in releasing your software.

Our basic assumption is that people are inherently interested in doing and learning good work. If you want the people in your teams to care about your product and about doing a good job, you have to first demonstrate that you care about what is important to them. From the outset of the project, we formed excellent rapport with everyone on the teams, based on mutual respect and understanding.

When we came to the project we found a general sense of apathy. It took a lot of relationship building and investment of time and positive affirmation to get many people back to a point where they cared about delivering personal value to the process, release management is a really important part of any IT organization and is not often given the attention it deserves.

There are lots of other great hints, tips and observations I can share about my experience of straightening out the release process of this medium-size enterprise. But these are the few most important for us in this particular case, though I suspect that they are pretty good ideas for any case.

Last but not least, a good release management process takes hard work, resolve and great communication; however, the greatest skill is the ability to review, learn and adapt different techniques over a period of time, because every organization is unique, every product, project, service delivery life cycles are unique, unless a release manager adapt a few techniques of his own he can’t provide a good release management process to his organization.

This kind of problems usually I have seen almost all my new companies, configuration and release managers are like an unsung hero’s in their respective field, until unless the project’s release activities are became worse, many organizations never think about the need of experienced (not knowledgeable) full time release managers, this kind of approach should change and need of specialist is very much required in any filed to reap their business.

--- End ---