Jan 19 · 7 min read

How To Run A Successful Software Development Process

Traditional product development is generally conducted in one "right" way, making it easy when it comes to predictability and reproduction. However, the process of software development isn't an exact science, where there is just one right way to do things. The software development process is a lot like art, where there are several different approaches to creating your product.

Non-technical leaders in the space don't have a good reputation amongst software developers. But you see, the key to running successful software releases is entirely non-technical. It's about the process. It's not necessary, but certain parts of a development process do benefit from technical know-how. Releasing software successfully into production is less a question of code or design alone and more one

of robust process architecture. Today we'll talk about getting the product from concept to production.

The value-driven statement

Before we begin, make sure your value-driven statement looks something like below:

Your product is a: explain what your product is.

That helps: Who are your targeted audience?

Solve: What problems do your target audience experience that this product solves?

By: How does your product solve it?

With: What is your secret sauce?

If you're basically developing a simple add-on for your business, some of this might be overkill. But when you're trying something new, this can help keep you remain focused.

The Roadmap

For their code, your team needs to assemble a clear roadmap. It must include a set of schematics, each fulfilling a separate purpose. For individual applications, these schematics are different. Application architecture diagram, user-interface mockup, and business process model are common. Technical expertise allows you to better assess your team's architecture and make sure they're on the correct path using these schematics. These schematics will be crucial even despite technical skill. When it comes to product completion, you can take their help to steer productive conversations. This way you will not have to best-guess at the "percent complete" from the development team. To figure out how close the product is to completion, you can track the status of each item on the diagram. And based on how quickly the team completed prior components, you can project future velocity.

Be meticulous about documentation

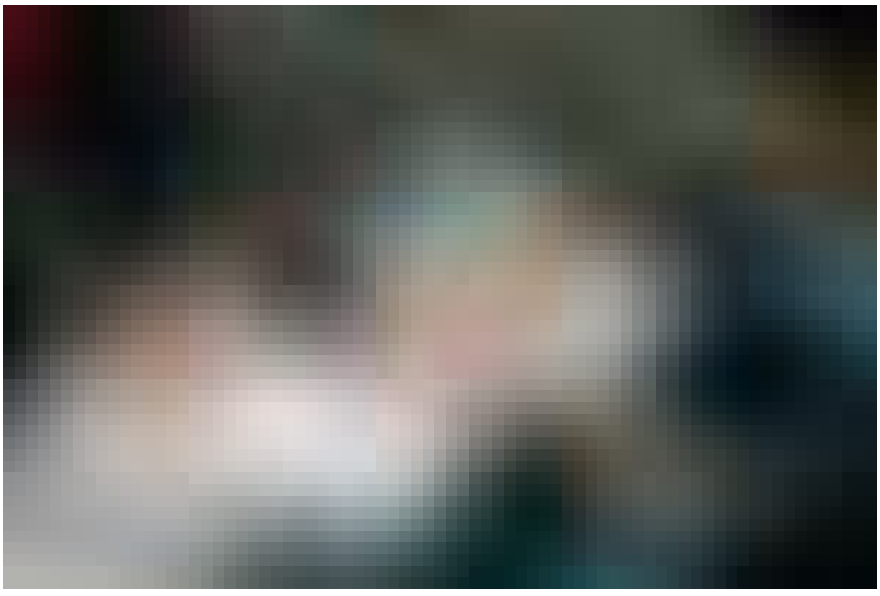
Documentation of processes is a method of scripting your team's behavior in a way. Especially if you're a distributed team, it's essential

that there's a place where your team can find what they need without needing to wait for other members in different time zones to come online and answer their question. Also, there is a wrong amount of pre-development documentation, none, but there is no right amount. Figure out what makes up a workable roadmap with your team before they sit down to code. In your development process, the first checkpoint will be to review this documentation and make sure they've met this agreement.

You don't have to reinvent the wheel

You need to make sure your team is focused on what they actually need to build. Begin by defining the key differentiators from products that already exist. Most of your team's effort and time must go in supporting this differentiation.

The schematics should come in handy here. Does your application include a signup and login process? A logging component? The point is for your team to use what already exists wherever possible. Then quickly get all the scaffolding in place so you can test your product. Then iterate through and without worrying about delaying production-readiness, change anything that helps differentiate your product further.



DevOps

Now the next checkpoint is to find out what part they are planning to build from scratch once you've reviewed the planned architecture.

Then identify the pre-built technologies you'll work with and go through these with the production support group.

Operations readiness will be the next checkpoint. A huge part of the secret sauce noted as DevOps is making sure the production support team is in the loop early on. DevOps is a belief wherein production operations teams and software development works together on common goals. The advantages include reliable code, quicker releases, and more time in developing because of automation. These benefits are all great, but they are a result of a strong communication process. Remember, automation is a collaborative effort.

Implementation and Testing

Here the soft skills demanded of leadership entirely eclipse any technical skill. Work with your implementation team to come up with a process for dividing work among themselves. There will always be a bunch of developers who want to work on all the interesting work and ignore any drudge work. They might argue that they're the smartest people and should hence get their pick of assignments. Some others would resist change and only stick to the same kind of work they've done before. An equitable distribution of work is what you should strive to lead your team into. Push everyone to grow appropriately and to collaborate and share.

Iterative delivery

Typically, in an agile development process, you will divide the implementation process into several checkpoints rather than a single deadline. They are called iterations. Refer to the roadmap you defined. And ensure what you've started is at least dev-complete, before starting new components. This reduces risk and gives you an accurate picture of the speed of development.

Push the code to an environment for acceptance testing as you complete the iterations. This involves test or pilot users who interact with the partial product. They test to make sure it meets design expectations and give feedback on how it could be better. But acceptance testing is not a substitute for unit testing. And you're ready to begin the release management process, once you accumulate enough tested code to make for a sufficient product release.

Code review

So now your team is convinced the code is done and acceptance testers are sure the product is working the way it should. The next checkpoint is to validate the belief that your code is ready to become a product. You may not be comfortable reviewing the team's code yourself if you don't have the technical know-how and that's alright. You won't have to. Your process should. Work with your team to come up with a process for code review which works for them. Establish a peer code review program by working across team boundaries. Using your schematics as a reference point ask them to explain how the code accomplishes the goals laid out in the schematic. By the end of the code review process, the reviewers and the developer must feel alright with being held accountable for the code.

This code review is also the perfect time to review security and documentation. Security review must find a place in any code review. This generally involves looking at the code once more to identify weak areas where a hacker could gain control of the server or use it to disclose private data. Your developer can take care of this if, even if they are just running a security code analysis tool that is automated.

The last checkpoint

The code has passed the review process. It's ready to be a product. But it still might not be ready for production. You have to clear the last checkpoint, deployment readiness. Is your code easy to deploy to production? This must include as little manual steps as possible. And in case the code doesn't work, you must have a plan to revert to the change as planned or the rollback plan. Your operations team should review the deployment and rollback documentation and let you know if it's enough. You can do this step yourself. But make sure the instructions for deploying the product are clear and simple. There should be very few manual steps since every manual step introduces a chance for human error. Once you've crossed this checkpoint, push that code into production.

Post-release

It's important to circle back and review how the process went once you're done, be it a success or failure. Did the testing rightly model the production scenario? Did your team correctly estimate the effort required to release a product? Review how well the team performed by revisiting the implementation and testing checkpoints. How is the product running in production? Maybe pay a visit to the operations team and gather their feedback. This will build trust between the two

teams, leading to more DevOps advantages down the road. Above all, make sure you are holding yourself and your team accountable. Your team will adjust their performance accordingly as they grow used to being held accountable for every step in this process.

Conclusion

> *A successful software release involves a well-understood, well-documented process for moving software through the pipeline from idea to product. You don't need extensive technical know-how for this. Often, technical knowledge can be a crutch.*

Make sure you are engaging with your team while plugging the holes and building a repeatable process that works for all of you. It needn't be perfect for everybody involved, but it must be understood by everyone. Also, make sure the demand for the product is matched by the velocity of your product through the checkpoints. And as with any process, iterate. Just like with the code, your first untested draft might not be perfect. Adjust the process on every run-through, and you'll finally have a predictable, smooth software release path.

. . .

Originally published on Product Insights Blog from CognitiveClouds: Top SaaS Development Company