More often than not, the beginning of a project catches you unprepared. You might be on the project team from day one, but the schedule is tight and there's not enough time for preparation. Even worse, you might overlook some steps, and that might come back to haunt you later.

The project kicks off, but after a few months you end up in a dark spot. Those annoying things that got you frustrated during your previous assignments are back.

If that sounds all too familiar, this article is for you. Follow these guidelines when starting a project and set yourself up for success.

Establish clear communication paths

Starting from day one, make sure that the roles are well-defined and everyone knows who handles what. People will request access to external systems, ask for clarifications, or signal emergencies. Whatever the scenario, make sure that the key contact persons are easily identified. Keep this information in a well-known location and make it accessible to everyone.

Define best practices and conventions

When the project kicks-off, don't start coding right away. If you do that, more often than not your codebase will become a tangled mess that nobody wants to touch or maintain.

Take some time to assess your past experiences and identify what went well and what didn't. It will help you define how you want things to happen for this new initiative. Involve the entire team in the exercise and make sure that you listen to what they're saying.

The end result should be a list of conventions and best practices validated by the entire team. Follow these conventions and you'll develop a much more well-organized and coherent solution.

Create a meaningful Definition of Done

How many times did you find out just before the demo that you can't showcase a feature because something is missing? I know, too often.

Developers tend to consider that a feature is done once it works on their local machine. However, a complete software development cycle involves much more than that.

The feature might only work on your local machine, so you have to test it on at least another environment. Then, you should ask your peers to review your work. This ensures that the acceptance criteria and quality standards are being met.

The next step is to add the deployment steps, so that you can release the feature to the demo environment. You should put these steps together in the Definition of Done, along with any other relevant step.

After that, make sure that your team uses the Definition of Done as a checklist before they complete a task. This will give you a lot more confidence that a resolved ticket is what you expect it to be.

Choose an appropriate continuous integration system

Continuous integration is crucial for every project. You want to make sure that you can release the new developments with minimal effort. Luckily, there is a wide range of options available on the market, Jenkins and TeamCity being two of them. There are a couple of very important factors to take into account when making this choice, though.

One of them is the preference of the team, and the other one is the price of the tool. It's always wiser to use a continuous integration tool that your team has used before. That means that everyone is familiar with the tool and you won't need to put in extra effort to learn a new system.

Don't overlook the pricing factor, though. If your tool of choice is a pricey one, the project sponsors might not be willing to pay for it. We all know that this happens, but it's not the end of the world.

There are plenty of powerful continuous integration tools that are free. Take the time to test them and choose the most appropriate one for you.

Choose your tools and applications

One thing that you want to avoid is using too many different tools for achieving the same purpose. Let's imagine that your team has to develop a REST API.

One of your less experienced team members, John, needs some help to test an endpoint for user creation. He asks Alex for assistance, who is eager to help, until she realizes that John is using SOAP UI to test his endpoints.

Alex, a big fan of Postman, spends a few minutes trying to understand how to use the tool, but without too much success. She gives up after a few tries, so they decide to reach out to George, the most experienced guy on the team.

However, George is an old-school programmer. He likes to do everything from the command line, so he asks John to rewrite his API calls to test them with cURL. It's not an ideal situation for anyone.

We know that developers love to have the freedom to choose their own tools, but it's not always optimal to work that way. In the long run, the team will benefit more from using the same toolset. Try to avoid this kind of situation by choosing specific tools for each purpose.

Don't be too rigid, listen to the preferences of your team, but make a clear choice. Also, make sure to explain to everyone why this is important and get the buy-in from your peers. After all, you don't want to be a control freak.

Use version control systems wisely

Version control systems are a must for every software project. You cannot develop a software solution collaboratively without using one. However, it's not enough to select a version control system and communicate the choice to your team.

You have to invest some time to define the way in which you want the system to be used for your new project. A good starting point is to compare the existing workflows and to decide what is the best one for your use case.

Next, you should validate the workflow with your team. If the feedback is positive, chances are that the version control system will be used as intended.

Avoid having multiple document management systems

One of the most frustrating things is when you need to find a piece of information and you don't know where to look for it. This usually happens because there are too many places where it could be. It shouldn't be that complicated!

When your DevOps guy wants to find the IP of the QA server, he should have to look into a single place. To make this happen, you should choose one good document management system and stick to it.

Keep it organized and confront anyone who attempts to store information outside the system. Everyone will see the benefits in the long run, even the people that get shouted at.

Define the environments required for your solution

We all know that developers, testers, and business should not use the same environment. But this aspect is one usually overlooked during the initial phase of a project.

You should start considering what environments are necessary for your projects early on. Getting approval and setting them up might take a while, so it's better to start as soon as possible.

As a guideline, you should have at least four environments: development, User acceptance testing (UAT), staging, and production.

Development Environment

The development environment will be the sandbox of the development team. That's why it won't be stable at all times, and you can expect data inconsistencies.

User Acceptance Testing Environment

The UAT is intended for user acceptance, so this is where the business people will do their testing.

Staging and Production Environments

Finally, the staging and production come hand in hand and they should mirror each other. This will ensure that the operations run on staging will have the same results on production.

But each project is different. You might need more environments than the ones above, or you won't be able to have them all due to the high costs.

Whatever the case, make sure you consider the system landscape upfront and define what you need so that you can deliver the project.

Prepare the codebase and project structure

A well-organized codebase will go a long way. Start taking a few proactive steps early on, so that your project won't soon turn into a big ball of mud.

You should define the main modules of the project, codebase structure, file naming conventions, packaging rules, and so on.

Make it as intuitive as possible, so that is easy for everyone to find the things they are looking for. Also, look back to the lesson learned from previous projects and make sure you don't repeat the same mistakes.

Create a document for local project setup

Even if you start with a very small team, chances are that you're not going to be the only ones on the projects. When new people are about to join, you want to make their life as easy and possible and help them get up to speed in no time. This has to start with the local project setup.

I've seen too many cases where the new guy has to spend an entire week to get the project running on his machine. This usually happens when the setup documentation was poorly written, incomplete, or missing altogether.

To avoid this, start with a document that defines all the steps required for the project setup. Then, make sure to test it on your own and refine it.