Agile Teams Care About Dev Ops

WRITTEN BY STEVE BERCZUK

While it's good that the idea of considering production issues early now has a name (Dev Ops), what matters in the end is delivering software more reliably, reducing waste in the product delivery process, and making stakeholders happy. Since product's don't deliver value until they are deployed and working it's good to take a step back and consider operations and deployment processes and team as first class stakeholders earlier.

Practice

Agile methods address the issues of complexity in technology and requirements by acknowledging that it's hard to design products and processes up front, and it is better to iterate in small steps and adjust based on your results. While much discussion of agile methods focuses on the development process, this approach also applies to deployment and infrastructure. To iterate effectively, you need to have repeatable processes that you can evaluate periodically and improve as you discover flaws. Software configuration management techniques are thus natural enablers of agile software development. Repeatable development processes can happen at many levels.

Repeatable workspaces and builds help developers collaborate effectively on a code level and give you practice building and running the application in a variety of environments. By having a private system build that allows anyone to build, configure and run product teams can quickly identify configuration and environment aspects that vary, and make the build and configuration mechanisms more robust. [Berczuk]. Continuous integration practices give you the ability to develop reliable integration testing practices, in addition to simply ensuring that the build works in a "canonical" environment. [Duvall]. The next logical step to improve product quality is to get more practice with your deployment practices and consider continuous deployment [Humble].

While automating and practicing your deployment practices will address some of the issues you might have with deployment quality, just executing the practice that developers implement is not enough, you want your system to have a deployment process that serves the needs of the key stakeholders in that process: the operations team. To enable this, you can benefit from considering what makes agile software development effective.

The Agile Difference

Agile software development methods seek to break down walls between the various parts of the traditional product development process. Rather than having distinct requirements, development, and testing phases for a project, agile teams divide the development process into iterations where the whole team works together as one. Agile methods do this by introducing project management and technical practices that allow for frequent, rapid feedback on the quality of the code and the requirements so that teams understand the state of the project all the time based on the artifact that matters most: working code.

The technical practices that are most obvious in an agile team are developer testing, continuous integration, and automation. Testing is no longer a hands-off process, reserved for a QA team. And by building the software (and running automated tests) after every commit you work to ensure that your code always build and works. In an ideal world, you can release your product at any point.

While continuous integration is a necessary precondition for software to deploy, it is not sufficient for deployment into a production environment if your deployment mechanisms, these mechanisms are development centric. To be successful at creating reliable deployment processes you need to extend your agile stakeholder model to include the operations team and their concerns.

On to Dev ops

Dev ops is an extension of agile[Cowham]. Most software developers have heard of continuous integration and unit testing (even if they do not use it). Most teams understand the value of versioning all source artifacts in a repository[Berczuk] and having easy to reproduce builds.

The next logical step is to routinize deployments into production. For this to happen the project team needs to incorporate operation issues into the backlog early, and incorporate operations team roles into the core development team.

Some concrete steps that teams can take to move towards a dev ops perspective are:

- Including the operations team in the project team. Some tasks might lend themselves to implementation by someone familiar with the operations systems.
- Extend your toolset to include tools that the operations team can easily use. Your may be developing a web application in Java, but some deployment and configuration tasks might be easier to execute in, say, Bash of Python.
- Developing tools and process to do deployments into production early. These tools should be in a form that works well in an operation environment, and be as automated as possible.
- Making realistic deployment environments available to the team so that production deployments can be exercised early, and automated.
- Place system artifacts under the same change management paradigm that you use for code. The tools may be different, but the you should strive to deploy not only an identical version of the code, but an identical version of the code in an identical server configuration.
- Each of these steps seems simple, but there are technical and organizational challenges to making them work reliably. In most cases the payoff will be worth the effort.

Why Dev Ops Matters

Teams are starting to realize that the development processes of agile, while essential to successful incremental deployments, are not enough for delivering value reliably. Operations is an essential part of the product value chain. Much as teams are willing to invest in build infrastructure such as maven repository management tools, teams are slowly discovering the value of considering tools for managing and automating deployments to production environments. Whether you call it Dev Ops, or a Larger Team Approach, Collaboration, or simply Doing the Right Thing, incorporating operations concerns earlier in the process will help you identify potentially critical issues earlier and address them with minimal cost.

References

[Berczuk]Berczuk, S. P. and B. Appleton (2003). Software configuration management patterns : effective teamwork, practical integration. Boston, Addison-Wesley.

[Duvall] Duvall, P. M., S. Matyas, et al. (2007). Continuous integration : improving software quality and reducing risk. Upper Saddle River, NJ, Addison-Wesley.

[Humble] Humble, J. and D. Farley (2010). Continuous delivery : reliable software releases through build, test, and deployment automation. Upper Saddle River, NJ, Addison-Wesley.

[Cowham] Cowham, Robert, (2010-October) Agility Throughout the LifeCycle – the Rise of DevOps http://www.cmcrossroads.com/component/content/13763

About the Author

Steve Berczuk is a consultant and developer who works with Agile teams. He has over 20 years experience developing application, often as part of geographically distributed teams. In addition to developing software he helps teams use Software Configuration Management effectively in their development process. Steve is co-author of the book Software Configuration Management Patterns: Effective Teamwork, Practical Integration and a Certified ScrumMaster. He has an M.S. in Operations Research from Stanford University and an S.B. in Electrical Engineering from MIT. You can contact him at steve@berczuk.com