Editor in Chief: **Diomidis Spinellis**
Athens University of Economics and Business, dds@computer.org

Being a DevOps Developer

Diomidis Spinellis

THE IDYLL has been going on for decades. DevOps, the synergy between software development and IT operations, was an open secret before it became a mass movement. Passionate programmers were often also closet system administrators—sometimes literally so, by nurturing recycled hardware in their home's closet. These same programmers were also drawn to the machine room, chatting with the administrators about disk-partitioning schemes, backup strategies, and new OS releases. Not to be outdone, zealous administrators would find endless excuses to develop all sorts of nifty software: deployment automation, monitoring, provisioning, and reporting tools.

Many factors are propelling the increased adoption of DevOps. First, software is increasingly being offered over the Internet as a service instead of being developed as an organization's bespoke system or a shrink-wrapped product. This makes operations an integral part of the offering, driving demands for service quality. Then there's the agile movement. Its emphasis on cooperation between all stakeholders has helped formalize the relationship between development and operations. Its acceptance of change has driven demand for processes and tools that will let systems respond to change swiftly and efficiently. Another enabler has been the availability of powerful and plentiful hardware. It has allowed the abstraction of system infrastructure and its expression as code amenable to established software development practices. Resource virtualization and cloud computing have provided the required building blocks.

In many IT sectors, DevOps is here to stay, helping deliver higher-quality services more efficiently. How can you, as a software practitioner, embrace DevOps to increase your organization's effectiveness?

// TODO

Start by cooperating more closely with your IT operations colleagues. Involve them in all stages of your software's development. Elicit their requirements to find which tools and APIs they need to deploy the system efficiently and to manage it effectively in production environments. Exchange views on architecture and features that will make your software more reliable, more scalable, and easier to deploy, configure, and run. See how you can issue software releases that painlessly mesh with running systems. Discuss planned changes and how they'll affect operations. Many old-style organizations have development and operations work in disjoint silos. Strive to break these down by instituting regular meetings, setting up shared (virtual and physical) workspaces, and embedding people from one group into the other.

If your application domain allows it, let agile-development principles guide your relationship with operations. Prefer to interact with your operations colleagues rather than be guided by rigid processes. Collaborate with them to solve problems rather than fight over service-level agreements. Focus on software that's running and delivering a service rather than on comprehensive documentation and formal handovers.

IEEE Software Mission Statement

To be the best source of reliable, useful, peer-reviewed information for leading software practitioners—the developers and managers who want to keep up with rapid technology change.

 0740-7459/16/$33.00 © 2016 IEEE

Help operations respond to events on the ground rather than hide behind an established plan. Remember: the operations team is one of your software's users. Framing your nonfunctional requirements as user stories aimed at the operations team makes ops a first-class stakeholder.

Expand the cooperation at the technology level. Start by extending your continuous-integration cycle to include the software's deployment on a test server. As you gain confidence with this, carry on with planning and practicing continuous delivery: the deployment of each software release to production. Add feature toggles and other infrastructure required to control your customers' experience and satisfy your business model.

First-class operations teams hate manual work (they call it toil, and they strive to eliminate it), so make your product amenable to automation. Ensure the software's deployment and runtime behavior can be easily controlled from the command line or through scripting APIs. Document these features and give them first-class status by designing any manual operation methods to work through the automation interfaces. Use file formats that other programs can easily parse and generate. Similarly, adopt control interfaces, such as REST (Representational State Transfer), that can be used with minimal ceremony. You don't want to bury your operations counterparts under five layers of abstraction and hundreds of dependencies.

First-class operations teams also hate flying blind. Alerts help them respond quickly when problems arise, while trends provide feedback from operations back to development. Equip your software with mechanisms that let others monitor its functionality and performance. Use a full-featured logging library, include logging statements in your software, and document the interfaces that control the logging verbosity. Write logging statements that the operations team can easily dissect, correlate, and aggregate to analyze your software's operational performance. If your systems support a whole-stack tracing tool, such as DTrace or LLTng, detail its use to scrutinize your software's operation. Provide ways through which system-monitoring watchdogs, such as Nagios plug-ins, can verify that your software is alive and well. For higher marks, provide information regarding your software's load and performance metrics, such as throughput, latency, resources used, and unserviced requests.

Finally, as you cooperate more closely with your operations colleagues, strive to learn from each other. You can help your operations team transplant into their setting your successful development practices, methods, architectures, and tools: how you use revision control tools to develop on multiple branches, how you document useful designs as patterns, how you program in pairs, and how you perform continuous integration. You can also learn a lot from the operations side: the relentless focus on service, quality, and reliability; the control of risk; the organization of complex deployments; the use of system configuration management tools; and the elimination of toil.

Thinking like a DevOps developer is an essential trait of an enlightened software professional.