A gentle introduction to **agile software development** and **DevOps** for non-technical people

A short tutorial by Kevin Matz <kevin.m.matz@gmail.com>
Version 1.2
18 February 2017

Are you a businessperson who needs to know what agile software development and DevOps are all about? This is a short introduction aimed at explaining these concepts to people with no background in technology or software development.

Software development is a technical process, and involves a lot of technical terminology, some of which is simply unavoidable. However, in this introduction, I will do my best to explain the terminology and ideas in in the simplest possible way.

Contents

Traditional software development approaches

In software development, the traditional general project management approach is called the "waterfall" model. The main idea is that teams communicate via "artifacts" (deliverables such as documents or pieces of software), and one team's outputs serve as the next team's inputs. There is a cascading "waterfall" of dependencies between the teams, typically something like this:

- **Business analysts** or **product managers** talk to the client or the customers to determine the requirements. In other words, they determine what the product must do. They then document the requirements and design how the product should work. The requirements and designs are written up in one or more specifications documents.

- **Software developers (programmers)** read the specifications and build the software (i.e., write the code) to build the product according to the designs.

- **Quality Assurance specialists (testers)** take the specifications and the software, and verify whether the product that was built actually matches the designs and meets the requirements.

- Technical specialists such as **systems administrators** or **release managers** then "release" or "deploy" the software. For a website or "cloud" application, this would involve installing the software or upgrades on one or more servers. (This could also take other forms: for a mobile app, this would involve submitting the app to the various app stores.)

- For complex products and major websites, various IT staff monitor that the system is running, maintain databases for optimal performance, and deal with emergencies and security incidents.

- **Technical support specialists** work with customers to resolve issues and help diagnose and fix problems with the product.

- Developers fix problems ("bugs") reported by technical support, and make minor improvements and changes. This ongoing process is called **software maintenance**.

- Major changes and new features are analyzed and designed by the business analysts or product managers. They prepare specifications for the next release of the software.

Traditionally, releases and upgrades are planned long in advance. Major new versions of software would be planned to be issued perhaps annually, or every six months. For some

products like the Windows operating system, three or more years can pass between major releases.

Some of the main problems with the traditional “waterfall” approach are:

- It presumes that it is possible to figure out and write down all the requirements completely. This is difficult -- if you’re building a custom software product for a client, and if the client is a big organization, the client’s staff can rarely figure out and reach agreement on what they really want, and getting the information from the client’s staff members inevitably becomes a messy political process of interviewing people, resolving confusion, contradictions, and conflicting opinions, and building consensus around a single solution design. If you’re instead building a mass-market product, assumptions are being made about what customers want. In either case, it’s extremely tough to get the requirements and an acceptable design right the first time.

- It also assumes that the requirements don’t change. Requirements often change during a project due to changes in the competitive environment, changes in laws and regulation, changing customer preferences, changing technology, changes in corporate or marketing strategy, changes in budget and available personnel, etc.

- Each team basically has to wait for other teams’ deliverables. Project managers have to plan and control the timing to ensure that no teams are sitting idle waiting for dependencies. One team’s delays can throw the whole project into chaos.

Agile methodologies

Agile project management methods like Scrum attempt to resolve some of these challenges by introducing concepts that reduce the planning and documentation overhead, and make it easier to quickly react to changing requirements:

- The release cycle is sped up; rather than spending six months or a year to release new versions of the software, new releases are issued much more frequently -- typically every two weeks, or once a month. These release cycles are usually called **sprints**. This lets teams focus on designing and building smaller chunks of functionality, it lets the QA team continually test the software, and it gets new versions of the software out to the client or customers faster so you can get feedback much sooner.

- The idea of trying to perfectly plan and design everything up-front is replaced with the idea of having a general vision and a vague long-term plan, combined with a round of detailed planning at the beginning of each release cycle.

- Rather than writing detailed design specification documents, the team maintains a list of to-do items. These are often called **user stories**, which are short, one-sentence summaries of features to build or bugs to fix. (It should be noted that the user stories approach is suitable for some projects and teams, and less suitable for others.) A list of user stories is called the **backlog**, and one of the tasks of the "**product owner**" role is to continually keep the backlog prioritized in terms of the importance and urgency of user stories required by the client or customers.

- At the beginning of each sprint, planning is done to determine the scope, i.e., which user stories the team plans to build in that sprint.

The goal is to get early working prototypes of the software ready for the client or customers to start using much earlier than was previously possible. Feedback from the client or customers can then lead to changes in prioritization for future sprints, and the creation of new user stories.

In theory, agile approaches shift the focus away from formal processes, detailed plans, and lots of documentation, and instead the importance of people and interpersonal communication is stressed in order to get things done. (Note however that the official Scrum methodology is itself a fairly strict process!)

DevOps

DevOps -- a term combining "Development" and "Operations" -- is a relatively new innovation that has evolved over the past few years. It is particularly suitable for very large organizations that build "cloud" software applications or run large-scale consumer websites such as YouTube or Netflix.

DevOps is not a single product or technology. It's a very general term that describes a way of structuring software projects.

One of the main ideas of DevOps is to relentlessly **automate** various tasks that exist during the processes of 1. software development (coding), 2. testing, 3. deployment, and 4. operations.

In the past, the software development (coding) and testing parts were considered to be part of the "software engineering" discipline, whereas the deployment and operations parts were considered more of the responsibility of the Information Technology (IT) discipline. Software engineering is about building systems, whereas IT is charged with keeping those systems running.

With the automation of various processes, however, there begins to be a merger of sorts between the two disciplines. The IT side starts developing programs of its own to automate many of its deployment and operations management tasks.

Before explaining DevOps, first we need to explain some more basic elements of the software development process:

- Developers (programmers) write instructions telling the computer what to do. These instructions are written in a programming language such as Java, Javascript, or C++, and the code that they write is referred to as **source code**. Source code usually needs to be **compiled** by a program called a compiler. The compiler takes the human-readable source code and converts it to a form that a computer can run (for example, on a Windows computer, the compiler would usually create a .EXE executable file that can be run).

- Software development teams use a **source control system** (such as Git or Subversion) to manage the **source code repository**. When multiple people are working on a project, they will often need to edit the same source code files. To prevent one person's work from overwriting someone else's work, the source control system and repository work like a library. In the past, developers would "check out" a file to work on, make their edits, and then "check in" the updated file. While a file was checked out, other developers could not make any changes. Today, most source control systems let multiple people edit the same file at once, and then conflicting changes can be resolved through a "merge" process.

- Software testing is usually done in one or more **testing environments**. A testing environment is a separate copy or "instance" of the software or website, together with a set of data, that is set up purely for testing purposes. The testing environment is separate from the "live" **production environment**, which is the actual copy of the software or website, with the real business data, which actual users can connect to.

Now, on the software engineering side, the following practices could be considered to be part of DevOps, although these techniques existed long before the term DevOps was coined:

- **Build automation** means that there is a system in place to take the code in the source control repository, compile it, and generate executable software completely automatically.

- **Unit testing** means that developers write short programs that test parts of the system at an internal/technical level.

- **Test automation** means that developers or testers create programs and/or record scripts to test the software at the user level (e.g., clicking on buttons and checking that expected results are presented on the screen).

- **Continuous integration** is a scheme where a system is set up to monitor the source code repository. Whenever a developer checks in a code change into the repository, then the automated build is automatically triggered, and all of the unit tests and automated tests are automatically run. If the developer's code changes cause any of the tests to fail, then the development team is notified immediately by an automatic e-mail, and the team's first priority should be to fix the problems (or update the tests) so that all of the tests are "green" (passing) again.

Now, on the IT side, some concepts and techniques that are associated with DevOps include:

- Increasingly, **virtual machines** are being used. A virtual machine is an "image" of a computer's disk contents, which can be run as a "simulation" of a physical computer. The virtual machine can be transferred and run on another server, so, say, ten small physical computers could be replaced by one bigger computer that runs ten virtual machines. Other than perhaps slightly slower performance, users have no idea that the service they are using is running on a virtual machine rather than on so-called "bare" hardware.

- Setting up, or "**provisioning**", a new server used to be a lengthy, manual process of acquiring a new physical server computer, installing the operating system and any needed third-party software, connecting it to the network, securing it, and configuring the server and software. With DevOps and virtual machines, the goal is to fully automate these process, so that an entire new virtual server can be created at the press of a button. This automation, and the automation of various other installation and configuration tasks, can be done using a combination of custom-written scripts, and tools like Puppet or Chef.

- Instead of buying physical server computers, the trend is now for companies to rely on cloud computing infrastructure such as Amazon Web Services or Microsoft Azure. Those providers run vast numbers of servers, and companies can basically rent computing power and storage space as they need it. While it is possible to "rent" a server and run a virtual machine with your software on it, it's also possible to build software to run on a cloud-computing platform that is designed for scalability, meaning that companies don't need to worry about outgrowing a server. If the computing capacity (e.g., the number of simultaneous website visitors or users) exceeds what a single server can handle, the cloud computing infrastructure shifts the load around to other machines dynamically.

- Installing and setting up a new release of a software application on a server, and performing any other upgrade steps (such as database conversions) to make it ready for customers to use, is called **deployment**. Deploying a release often used to be a

time-consuming and very error-prone task, but the goal under DevOps is to turn this into a fully automated process.

- A system can be set up so that whenever a developer checks in a code change into the repository, the code can be automatically built, the automated tests can be run, and then the software can be automatically deployed into a testing environment. Quality Assurance staff can then manually test the software in this environment. If they are satisfied with the quality and stability of the software, they can then press a button to automatically deploy that version of the software into the live "production" environment for actual customer use. Rather than the six-month or one-year release cycle, at some companies, multiple releases to production are done per day, and this is called **continuous delivery** or **continuous deployment**.

  While releasing to production frequently would normally be seen as extraordinarily risky, now with carefully-structured testing procedures and the DevOps infrastructure that makes it possible to "**roll back**" the production software to a previous version in case of a problem, it is very much viable. Continuous delivery is seen as a way to be responsive to the customer: bugs and problems can be addressed very quickly, and new features can be rolled out at a much faster rate. Continuous delivery also allows for experiments and marketing tests such as **A/B tests** to be run on a temporary basis.

There is of course much more to DevOps, and the term DevOps encompasses an entire way of thinking. It is a very dynamic field where ideas and technologies are constantly changing, and where definitions are subject to debate. It's also a field where every company does it in their own different way. Therefore I'm sure would be easy to find people who disagree with the description I've given here.

In summary, DevOps is a general term that refers to technical processes and techniques for automating some of the tasks in software development, testing, deployment, and operations. DevOps makes it easier and more reliable to get software changes released to customers. For marketing purposes, it allows software and web companies to be more responsive to customer and market needs. ■