Agile Methodology and DevOps in a Cloud Environment

Don Filantrisec

Developer Cloud Research

Abstract

Software developers know that process is key to efficiently building quality products that can be consistently replicated. Methodologies such as Agile project management and DevSecOps (development, security, and operations) birthed the idea that in order to meet overall development and infrastructure goals, the industry must find a way to include continuous integration, delivery, development, and improvement throughout its activities. This is a look at Agile methods, DevOps, and cloud computing to discover what each brings to the table.

Agile Methodology and DevOps in a Cloud Environment

Software developers always knew that process is key to efficiently building quality products that can be consistently replicated. Methodologies such as Agile project management and DevSecOps (development, security, and operations) birthed the idea that in order to meet overall development and infrastructure goals the industry must find a way to include continuous integration, delivery, development, and improvement throughout its activities. Let's look at Agile methods, DevOps, and cloud computing to discover what each brings to the table.

Agile Methodology

At the time of its inception, Agile methods represented a groundbreaking shift in the way that software was brought to market. Prior to Agile, customers would present software development firms with an initial set of user requirements. They often didn't know what the finished products were supposed to look like, but they felt like they would know the envisioned product when they saw it.

The go-to software development methodology of the moment was the Waterfall model. When using this model, developers followed a sequential roadmap for developing software that included these five phases: requirements, design, implementation, verification, and maintenance. These phases had to be completed sequentially to fulfill contractual obligations with customers and to deliver quality products at project close out. With the Waterfall model, software developers spent a lot of time producing documentation, but the finished products often displayed underwhelming performance, were late to finish, and came in over budget.

The reason that the Waterfall model stuck around as long as it did was because it worked for certain projects. The Waterfall model was great for small projects that had set, well-defined requirements. However, the world of Information Technology (IT) was evolving at a fast clip, and efficient software was needed to power that technology. As a result, software projects became more complex as developers ran to take on new challenges and meet the demand of their industry.

If you've ever worked in the IT industry, you realize that software developers are a rather vain group of professionals. They knew that they had the expertise to build innovative products and wouldn't stand for constantly looking like failures when software development projects tanked due to cost and schedule overruns. The group came to the consensus that it wasn't their collective lack of talent that doomed hundreds of software projects to failure around the world. The process must be the culprit. Just because they are vain doesn't mean they weren't right. The Agile Manifesto was born.

Agile Manifesto

What are the core principles of Agile methodology? They are essentially these four tenets of the Agile Manifesto.

#1 Working Software Over Comprehensive Documentation

It turns out that the challenges of the new software development environment had completely met their match with the 1990s software engineers. As a rule, software developers were trained to think critically in order to solve problems. The first pain point that they identified and addressed was one that had to do with the internal steps that they took to develop software. A few of the engineers recognized that the code that they built as a hobby was developed differently and sometimes more efficiently than the programs that they developed within teams at work.

Their side hustle programs were developed as working prototypes. As they dreamed of additional capabilities, the engineers would add those functions to the prototype. After several iterations, they would have a satisfactory finished product. In addition to getting a working product that met their expectations, they were able to achieve the desired results faster because their project didn't require large amounts of documentation.

The software engineers who created the Agile framework made prioritizing working software over comprehensive documentation a core tenet of the Agile Manifesto.

#2 Individuals and Interactions Over Processes and Tools

The Waterfall model from which the engineers worked was heavy on process and light on human interaction. When engineers followed the process of requirements, design, implementation, verification, and maintenance in that order, they often missed out on important input from team members who should have been allowed to contribute earlier in the project.

When following an antiquated Waterfall process, engineers would work for a set period on requirements. At the end of the period, they would present their work at a requirements review. If gaps were found in their presentation, they would go back and rework the requirements. They had to get milestone approval to move to the design phase. While a cadre of analysts reworked requirements, other software professionals worked on other projects until their expertise was needed later in the project.

The engineers who drafted the Agile Manifesto believed that the work could be done more efficiently and with better results if a small, dedicated team worked on the project throughout all of its phases. The concept of self-organizing teams stemmed from the idea that team members could decide when to step in and pick up activities to work on as needed. Who knows how to run a software development project better than those people who do the work within the team?

Based on the above findings, the software engineers decided to make prioritizing individuals and interactions over processes and tools part of the Agile Manifesto.

#3 Responding to Change Over Following a Plan

Engineers found that an inflexible plan was the nail in the coffin for software development projects. However, planning is an inherent part of project management. Without a plan, you wouldn't know when to bring in workers to complete tasks or when to stop their activities to avoid labor cost overruns. The engineers decided to keep the concept of project plans but streamline them to make them more responsive to changes.

As the demand for more robust tech tools outpaced customer subject matter knowledge, software engineers were increasingly called upon to update requirements mid-project. Many of the new requirements went beyond the original project's scope. This was called scope creep, and it became an inevitable part of doing business with public and private sector customers.

Scope creep was a particularly worrisome problem because it impacted cost, schedule, and performance. Under the Waterfall model, customers had to wait until the end of the design phase to get a glimpse of how their software would work. When they noticed that a certain update was needed, it was too late to change the design without experiencing schedule and cost overruns.

The best-case scenario is that customers would decide to proceed with the current design and get the updated functionality within a later software build. The worst-case scenario is that the manager assumes the risk of overruns and attempts to build in the update. Many projects have been canceled because they ran out of money. Either scenario leaves customers less than satisfied and software development teams scrambling.

It was clear that the engineers needed a way to respond to change without being tied to an ironclad plan.

#4 Customer Collaboration Over Contract Negotiation

Software development firms frequently did business with the government during the late 1990s and still do to this day. Getting government contracts was lucrative but fulfilling them was often complex.

When government contracting officers believed that they had a pretty cut-and-dried project, they proposed a firm fixed price (FFP) contract. In a FFP contract, the software development firm estimates the cost of the project and delivers a bid to the customer. If it's awarded the contract, the software development firm is guaranteed to get that amount of money for the bided scope of work. The burden is on the software development company to bring the project to a close within budget.

While government contracting officers usually avoid scope changes at all costs with FFP contracts, requirements change that fall within the original scope of work can also be a challenge to implement. The engineers who drafted the Agile Manifesto believed that they could build a

framework that allowed them to exceed customer expectations through collaboration and not through time-consuming and costly follow-on contract negotiations.

Implementing Agile

Iterative Software Development

What is an Agile methodology? An Agile methodology is a process that incorporates all four tenets of the Agile Manifesto into software development efforts. There are quite a few of them out there.

While all of them use different approaches to achieve the Manifesto's goals, they have many things in common. One of those things is prioritizing working software over extensive documentation.

Scrum is an Agile technique that allows engineers to produce a piece of working software at set iterations. The team starts with a prioritized backlog of work activities. Higher-priority items represent functions that are absolutely needed to make the software work in its most basic form. This basic piece of working software should have all the functionality of the final product but in an abbreviated form.

Scrum co-founder Ken Schwaber likens the piece of working code to serving up a slice of wedding cake. You wouldn't serve a guest a cake layer or even a piece of a cake layer. You would slice down the center of the cake and capture all the yummy parts of the confection from its marzipan icing to its moist flour-based interior.

For certain Agile projects, development teams and customers agree that engineers will only make available the documentation that was needed to build the software using this iterative development method.

Effective Communication Over Heavyweight Processes

While software development teams know the important role that process and tools play within software development projects, many don't realize that less is more. Complex processes and tools slow development teams down and hinder meaningful conversations among team members.

Kanban is an Agile method that employs lightweight processes and tools to deliver the latest software products to market faster. "Kanban" is a Japanese term that means visual signal. In this methodology, prioritized work activities are printed on cards and placed on a board. Besides having a brief description of the work item, the card will contain the estimated time to complete the task and the name of the point person for the task.

Work items are displayed in distinct positions on the board, which include "To Do," "In

Progress," "Code Review," and "Done." Work activities that stay in the "In Progress" stack for too long are flagged, and the team rushes to help resolve any issue that's causing the slowdown. Kanban's transparency and simplicity allow software development teams to focus on communication rather than closely following processes and using complicated tools.

Getting the Job Done When Plans Change

The scrum methodology is a good example of how software engineers use Agile to respond well to changing requirements. Scrum projects are divided into two- or four-week sprints depending on the scope of the project. The typical roles for a scrum team include product manager, scrum master (process manager), and the development team.

The product manager pitches the proposed product to the customer who approves or requests revisions to the proposal. After receiving customer approval, the development team begins to work at a fast clip for the duration of the sprint. While the team is working and the scrum master is making sure that the team sticks to the scrum method, the product manager delivers updates to the customer. If the customer thinks of additional functionality that should be included in the product, the product manager will add the requirement into the backlog. The next sprint will reflect the modification.

By managing change in this way, updates to customer requirements are fully addressed earlier in the project. The customer gets informal updates about the product from the product manager during the sprint. He or she also gets an early look at the slice of working software at the sprint review, which is held at the end of each sprint.

That kind of transparency often spawns more refined requirements. However, development teams are spared constant disruptions since the product manager must wait until the sprint ends to introduce new requirements into the backlog.

DevOps Agile and Scrum

DevSecOps is the collection of tools and know-how that allow development, security, and IT operational teams to work together seamlessly. How does DevOps use Agile? Let's look at the commonalities between the two processes and how they work together.

If you're waiting for DevSecOps to preempt Agile, the experts say that you'll be waiting a long time. While Agile methodologies have pretty much replaced the Waterfall method in modern software development shops, DevSecOps isn't likely to do the same with Agile.

Agile and DevOps share many common elements. Here are the most prominent.

Reliance on collaboration to deliver products that exceed customer expectations is one of those shared characteristics. Waterfall software development projects attempted to capture a full set of written-in-stone requirements straight out of the gate. Armed with those specifications, neither

the project manager nor the lead engineer needed to talk to the customer until the next milestone review meeting.

For Agile projects, product owners are encouraged to keep communication lines with customers open. Customers are able to give feedback about working software prototypes at shorter milestones, which are called sprint reviews. DevSecOps also works optimally when practitioners get feedback from team members and external stakeholders. Both Agile and DevSecOps practitioners equate collaboration to higher levels of performance and product quality.

IT products and services that are slow to market aren't tolerated. Companies invest in the development of tech tools and platforms to make money from being the first to market with those products. When developers are late, business customers lose. This problem is particularly bad because technology is known to move quickly. A competitor company may roll out a product that has a richer feature set while your product is stuck in development. Both Agile and DevSecOps use techniques to speed up development and IT operations.

Those who are unfamiliar with the way that DevSecOps came to be may ask, "Can DevOps work with Agile methodologies?" DevSecOps was introduced to the IT community at the 2008 Agile Conference, and it was intended to be an extension of current Agile practices. For instance, DevSecOps uses Agile techniques to roll out operational platforms iteratively. This supports the continuous integration/
continuous delivery mantra of DevSecOps.

Cloud Computing

Cloud computing is the delivery of server, software, and databases virtually. Its use has become very popular in recent years. How is cloud computing boosting Agile software development? The answer is with speed and flexibility.

Today's software development teams may require hundreds of servers to develop, stage, test, and deploy their products. Besides being very expensive to purchase, physical servers take time to acquire and set up. By using cloud computing, development team members can provision hundreds of virtual servers in minutes without the help of network engineers.

Members of today's development teams are frequently located in different geographic locations. This makes practices such as pair programming and collaborative development very difficult. Cloud computing comes to the rescue with virtual development environments that are accessible to all team members on a round-the-clock basis.

Conclusion

The future of software development, IT security, and operations combines the best of Agile and DevSecOps practices to get high-quality products into the hands of users as quickly as possible. With improvements in cloud computing, there's no going back to status quo software development or IT service deployments.

References

Bose, Reekchhanda 29, July, 2021 *What is Waterfall Development?*

Kannan, Nari 20, August 2012 6 Ways the Cloud Enhances Agile Software Development

Buchanan, Ian Agile and DevOps: friends or foes?

12 Principles Behind the Agile Manifesto

Watts, Stephen 15, April 2019 The Role of Cloud in DevOps

Petkar, Prachi Introduction to Agile DevOps