What is Application Lifecycle Management?

David Chappell

Sponsored by Microsoft Corporation

Defining application lifecycle management (ALM) isn't easy. Different people (and different vendors) take quite different perspectives. Still, ALM is important, and so understanding what it encompasses is also important.

It's common to equate ALM with software development. Yet this simple approach is too limiting; ALM is much more than this. In fact, an application's lifecycle includes the entire time during which an organization is spending money on this asset, from the initial idea to the end of the application's life. To be both accurate and useful, our view of application lifecycle management should take an equally broad perspective. Anything else just isn't right.

It's also important to take a modern view of ALM. In a world where applications can run in corporate datacenters or in the cloud or on a mobile device or somewhere else, ALM needs to encompass a range of development processes. There's no single process that's right for every situation.

The Three Aspects of ALM

ALM can be divided into three distinct areas: governance, development, and operations. **Figure 1** illustrates this, showing each of these three aspects on its own horizontal line.

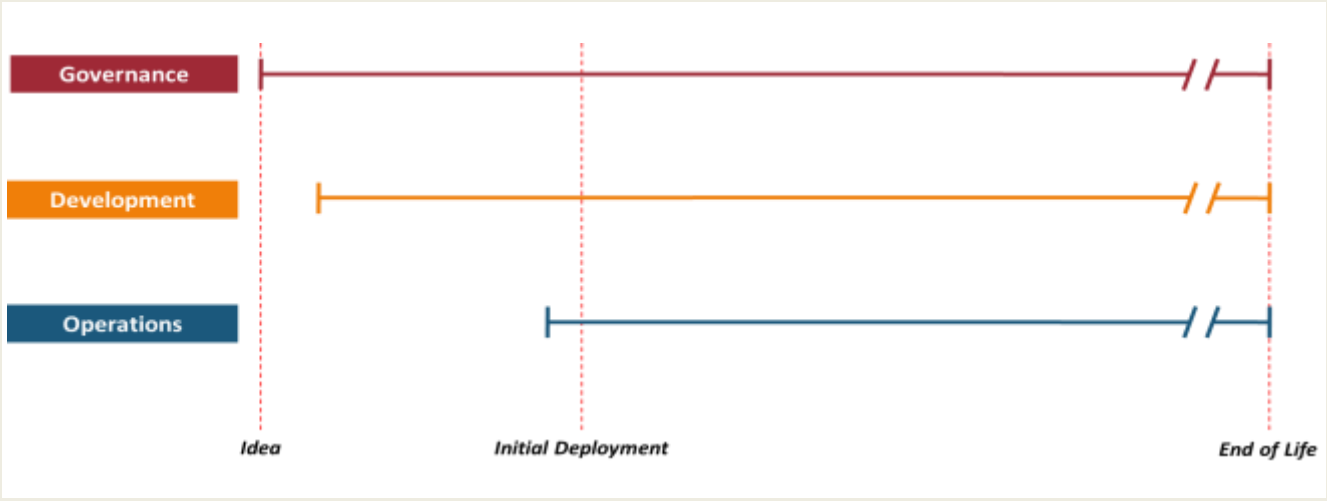

Figure 1 ALM can be viewed as having three aspects.

> *Like a human life, an application's lifecycle is demarcated by significant events.*

Like a human life, an application's lifecycle is demarcated by significant events. It begins with an *idea*: Why don't we build something that does this? Once the application is created, the next big event is its *initial deployment*, when the application first goes into production. And finally, when it no longer has business value, the application reaches *end of life* and is removed from service.

The three aspects of ALM—governance, development, and operations—span these events. Each aspect focuses on specific concerns, as **Figure 2** shows.

Governance, which encompasses all of the decision making and project management for an application, begins with business case development. Does it really make sense to build this application? If the answer is yes, the project begins, and governance now moves to project portfolio management. Once the application is initially deployed, governance shifts to application portfolio management, where the people paying for the application decide when (and whether) to enhance it and when the application has reached end of life. Governance extends throughout the entire ALM time span, and in many ways, it's the most important aspect of ALM. Get it wrong, and you won't come close to maximizing the application's business value.

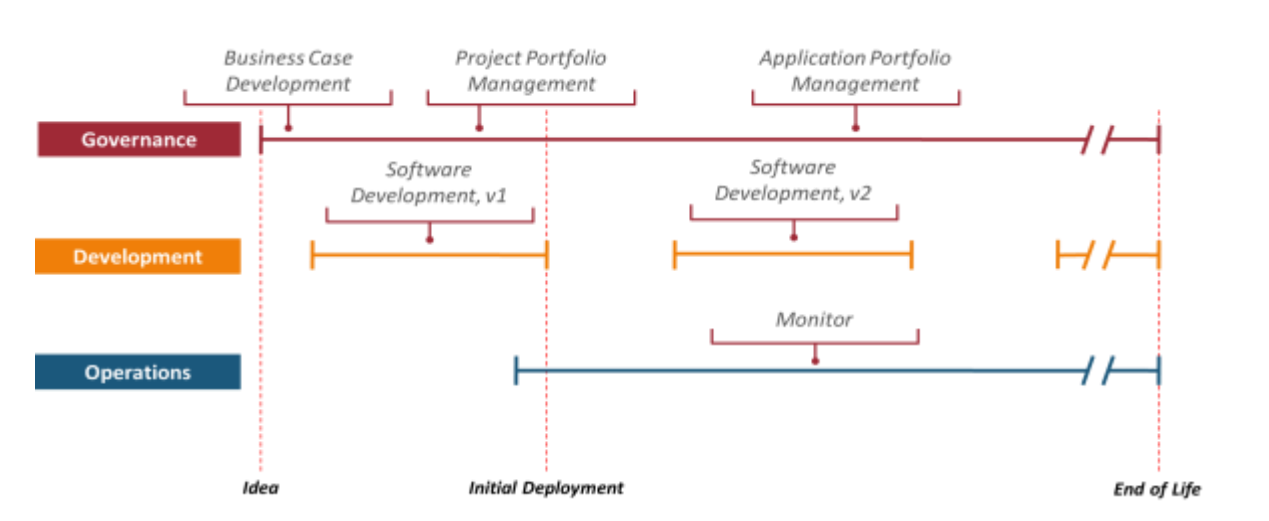

Figure 2 Each of ALM's three aspects has its own components.

ALM is much more than just writing code.

Development, the process of actually creating the application, begins when the business case is approved. As the figure shows, the development process can reappear one or more times in the application's lifetime as new versions are created.

Operations, the work required to run and manage the application, begins shortly before deployment, then runs continuously until the application is removed from service. This aspect of ALM is responsible for releasing the application into production, monitoring it while it runs, and more.

A key point to understand is that ALM is much more than just writing code. Viewing ALM as synonymous with software development is wrong—it leads to a misunderstanding of what's really required to be successful in this area.

ALM Styles Today

ALM always includes governance, development, and operations. There are variations in what happens on each of the lines, however, especially with development and operations. Three main approaches are in use today: waterfall, agile, and agile with continuous delivery. All three are worth looking at.

Waterfall development is the classic approach; it's how pretty much everybody once built new applications. This ALM style breaks the software development process into three sequential steps:

- *Requirements*, working out in detail what the application should do.
- *Development*, writing the application's code.
- *Testing*, examining the newly written code to find and fix bugs.

Once these three steps are done, the application is released for its initial deployment. When a new version is needed, the same steps happen in the same order: requirements, development, testing, and release. **Figure 3** shows how this looks.

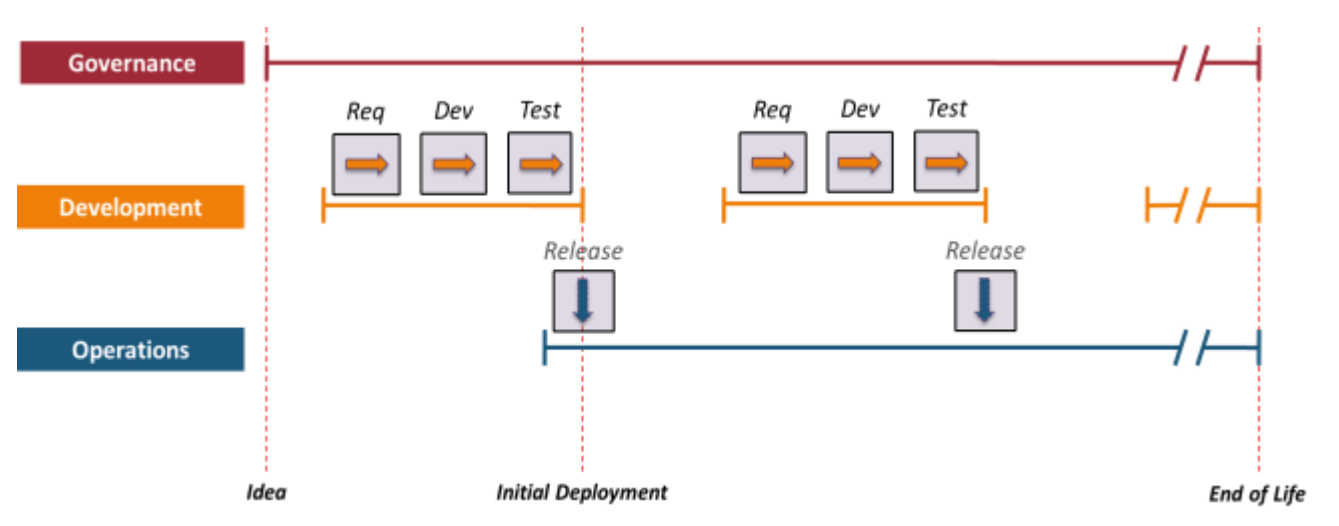

Figure 3 Waterfall development gathers requirements, writes code, and tests code, then releases the resulting application.

Waterfall development can work, and it's the right approach for some applications. Think about a team creating a new operating system, for example, that's released to its users once every two years. In a scenario like this, the waterfall approach might be appropriate.

But waterfall development has some well-known problems. Perhaps the biggest is that it assumes the development team can create a complete set of correct requirements before development begins. In many projects (maybe even most), this isn't possible—the requirements are bound to change during the process. In situations like this, agile development, shown in **Figure 4**, is likely to be a better choice.

With an agile development process, the same steps happen as in waterfall: gathering requirements, developing the software, then testing the result. Instead of going through this process just once, however, an agile process does them multiple times, creating many short iterations of the requirements/development/test cycle.

Because requirements can be added, removed, and re-prioritized at the beginning of each iteration, an agile process can accommodate change during the development process. Rather than require that all requirements be correct up front—something that's often impossible—agile development makes it easier to incorporate new requirements as they appear. Because of this, agile processes, such as Scrum, have become widely used.

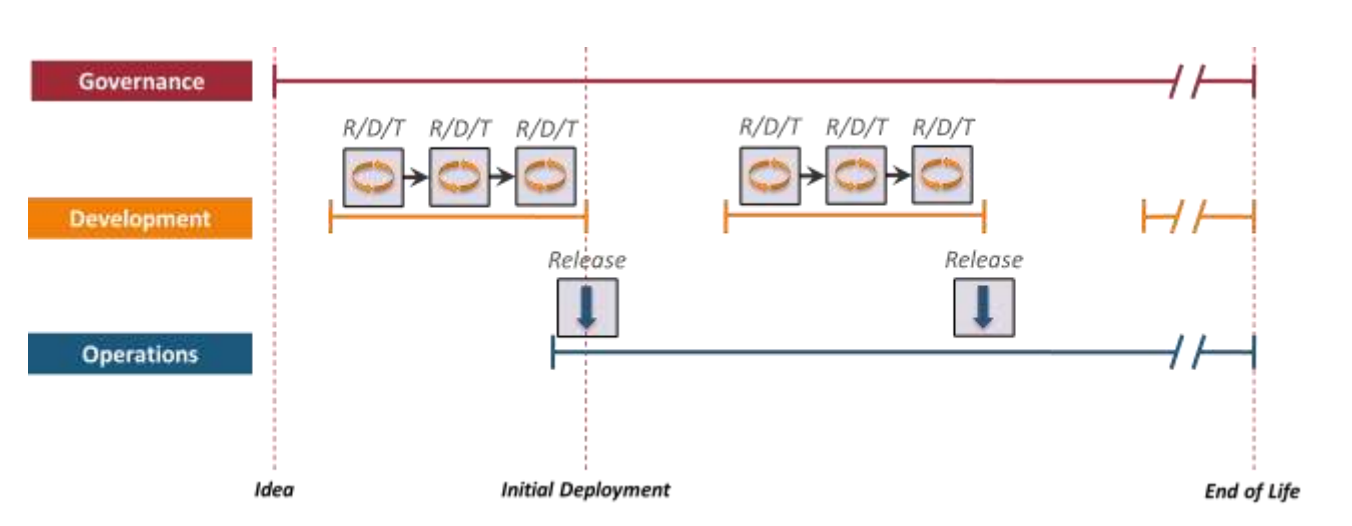

Figure 4 Agile ALM does iterative development, then releases the resulting application.

As the figure shows, however, agile processes traditionally deploy an entire application once it's completed. In many situations, such as an enterprise deploying a rarely changed business application or a software vendor shipping a product in a box, this can make sense.

In the modern world, though, this kind of infrequent deployment is no longer required. One of the tenets of agile development is that the development team creates a group of completed, tested, ready-to-run features in each iteration. Given this, why wait until the entire application is done to release it to operations?

Think about a mobile app, for instance, purchased from an app store that can deliver updates every week or two. Or suppose the application is a service running on a cloud platform. The creators of this cloud service can update its code whenever they like. In situations like these, releasing updates more frequently would let the application's users see improvements as soon as they become available. With the right tools, even a custom enterprise application running in a corporate datacenter can be updated frequently.

Rather than wait until an entire new version of an application is complete, why not release new features after, say, every iteration? This is exactly what's done by agile development with continuous delivery. **Figure 5** shows how this looks.

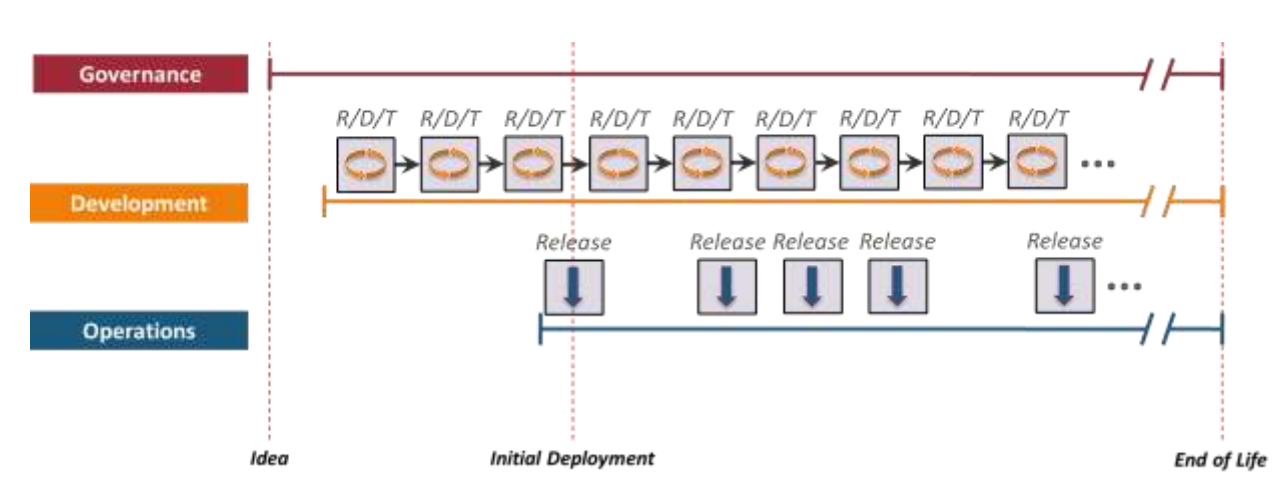

Figure 5 Agile development with continuous delivery does ongoing iterative development, then releases the resulting features regularly.

Continuous delivery doesn't necessarily mean that the result of each iteration is released to operations. It does make this possible, however, typically through automated tools. In fact, the frequency of release is driven by business requirements, not technology. Whether the new features created in an iteration get deployed is generally a business decision.

"The frequency of release is driven by business requirements, not technology."

When an organization can get value from making new application features available as quickly as possible, agile development with continuous delivery is the right choice. As long as the tools exist to do frequent releases, this approach to ALM is most likely to create the right software, then get it into production as quickly as possible. This lets an organization get the most value for the money they're spending on an application.

By themselves, though, tools aren't enough. As Figure 5 suggests, continuous delivery implies effective cooperation between development and operations people. Commonly referred to as *DevOps*, this kind of cooperation takes effort to create and maintain. Yet when frequent deployment of new features has real business value, the work of building a DevOps culture can be well worth doing.

Conclusion

ALM is much more than just writing code. All three aspects—governance, development, and operations—are tightly connected to one another. Doing all three well is a requirement for any organization that aspires to maximize the business value of custom software. But this isn't an easy goal to achieve. Each of the three is challenging to get right on its own, and so getting the combination right is even more challenging.

Think about a project that gets the initial governance aspects wrong, for example, perhaps by not understanding the business needs or failing to get the right stakeholders involved. No matter how well the organization does development and operations, this project won't provide much business value. Similarly, a project that targets the right problems using a first-class development process might ignore operational issues, such as providing enough resources to run the application reliably. Once again, the business value this investment provides won't be as large as it should be. Taking a broad view of ALM can help organizations avoid problems like these.

Similarly, choosing the right development style might take some thought. While waterfall can sometimes make sense, agile development has become the norm in many organizations today. And whenever an organization can benefit from deploying new features quickly, agile development with continuous delivery is an even better choice. As the tools and processes needed to support this approach spread, expect this to become the new default in the not-too-distant future.

There's no way around it: Taking a broad, holistic view of ALM is essential for success with this critical business process. Nothing else works.

About the Author

David Chappell is Principal of Chappell & Associates in San Francisco, California. Through his speaking, writing, and consulting, he helps people around the world understand, use, and make better decisions about new technologies.