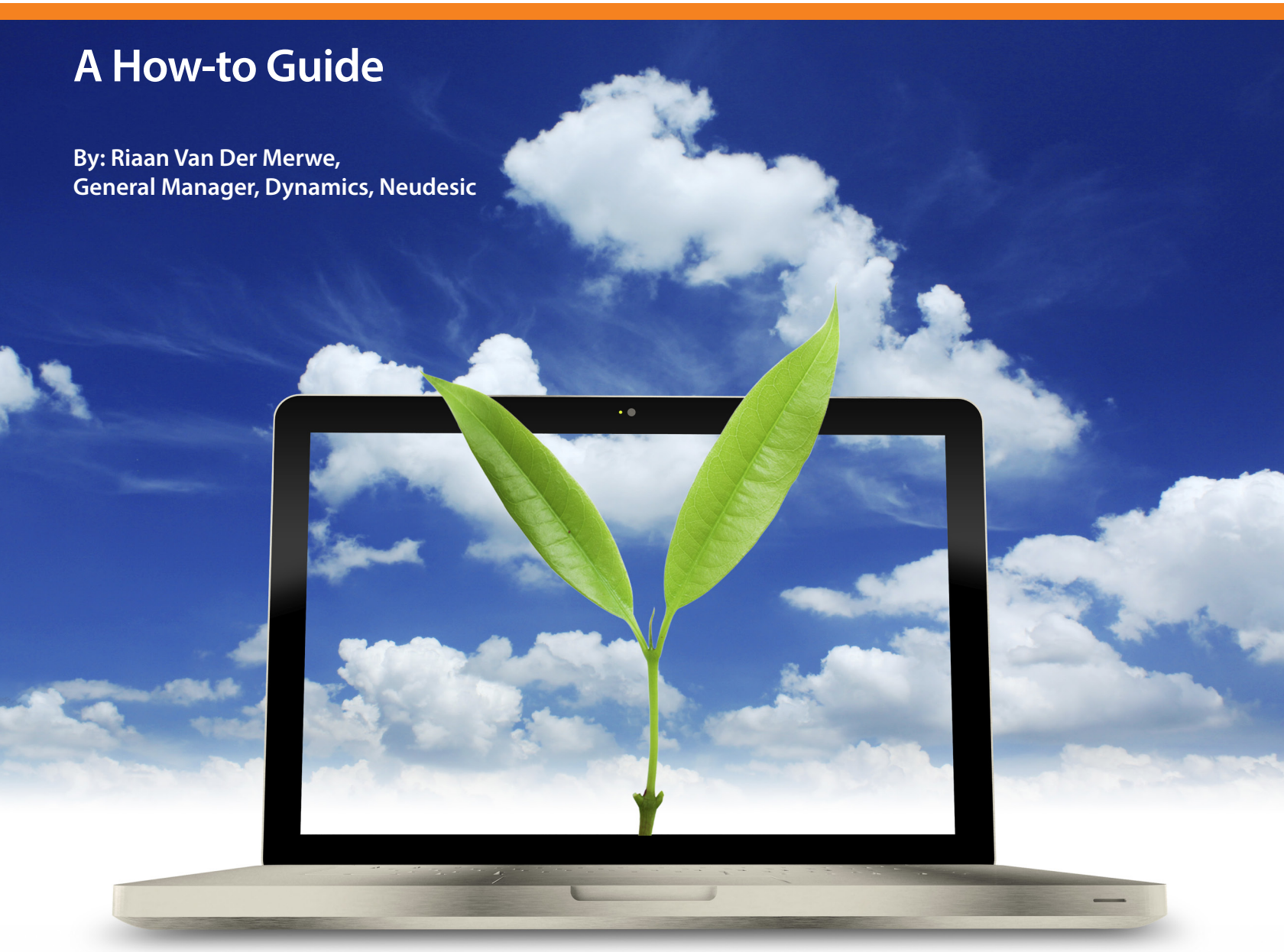
Managing Dynamics CRM 2013 Applications from Cradle to Grave

A How-to Guide

By: Riaan Van Der Merwe,
General Manager, Dynamics, Neudesic



NEUDESIC

Table of Contents

Introduction

Application Lifecycle Management, or ALM, is one of those processes with myriad variables that depend on a number of factors. It is also a process that, when done well, can streamline the entire lifecycle of your Microsoft Dynamics CRM application.

ALM is not a one size fits all process though. With Dynamics CRM 2013 in full effect, it's important to extract maximum value from the get-go. We've cut through the clutter to hone in on the most effective methods for deploying ALM in this environment. There are a number of choices to be made – not all are necessary for successful deployment but each should be evaluated, and then mixed and matched based on your specific enterprise, situation and goals. This blended approach to ALM offers flexibility and a custom fit to your particular needs.

Creating the Right Fit

If you look at the high-traffic sites that are currently prevalent – Facebook, Google+, LinkedIn, Twitter – they all build and test continuously, demonstrating quality and implementing fluid change. These entities are not stuck with big, single monolithic deployments, which can actually be more costly. It's so much harder to manage significant changes compared to a lot of little trickle modifications. What's more beneficial is constant, continuous integration: you're always going to be making mistakes, but you'll also be testing and amending.

Environment plays a key role in determining which ALM methods to deploy for optimum benefit. Your selection should depend on where you are in your project cycle, your team and the rigor of your enterprise. Large enterprises generally take a comprehensive approach to rigor, deployment and change management which affects ALM deployment very differently than it would a small organization. The level of your lifecycle choices depends primarily on what your company, or the project, is ready for and what the team can handle without putting the project at risk.

By weaving together the right approaches, you can build a strategy that works – both short and long term – to meet the needs of your enterprise. The following provides a comprehensive view of the various best practice approaches in ALM.

Solutions Strategy

In CRM, "solutions" are really packages, code packages that allow the export and import of application configurations. They also include code 'bits' that have been deployed into that particular application. Through combine filing, projects can be moved around for greater flexibility.

When developing a solution strategy, determine whether you are going to build a **single**, monolithic solution, or different **solutions per version** or component. Keep in mind, if you take the monolithic solution approach and someone makes a change, everything will be affected. Instead of taking the easy route, it's generally better to build one solution per set of functional requirements. This allows you to independently make version updates to specific functionality, and to layer deployments. By building one solution per component or fix, you can layer each separately, on top of one another. This practice meshes well with the tendency for people to work independently on their particular change. Having a very specific set of component fixes makes better sense and offers the best results.

But the 'single solution' versus 'solutions per version or component' is not the only consideration in a solution strategy. You must also determine whether to take an **unmanaged** or **managed** approach. Unmanaged solutions are applied to the base solution. You can't uninstall an unmanaged solution. With a managed solution, you can install and stack new versions on top of one another. The unmanaged choice works well when you're a small to midsized company – it's simple and straightforward – but it doesn't allow for much variance. With a managed solution strategy, greater control over version numbers and processes are required; this in turn allows you to equip your support team with the information necessary to know where you are within the process at the moment you issue a release. This makes it easier to build a baseline.

It is highly advantageous to use the core components for unmanaged solutions and then provide to the team as managed solutions. This offers a layering effect in which you can remove and augment, making specific fixes within certain layers. It becomes a 'cake with different slices' – with various components within the slices. If you have to perform fixes later on, managed components in that specific slice can be changed,

whereas unmanaged solutions leave everything open. You require much less rigor in your processes, yet it's easy for people to change things. If you want to perform fixes, you've always got to do them over the top.

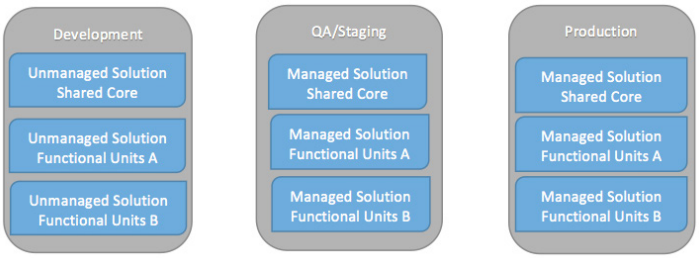


Figure 1

Using 'versioning' in the unmanaged solutions to the managed environments approach allows for bug fixes and enables new versions to be distributed readily to QA and Production. It is important that you never customize in QA or production. Rather, perform all changes in development and then deploy the managed packages to QA and production.

If you ask most CRM developers for their recommendation, they would tell you to just focus on the unmanaged approach, steering clear of the managed method. But the latest fixes in CRM 2013 provide a clearer path to using both a managed and an unmanaged approach for any deployment environment.

Development Infrastructure

You must also understand how your people are developing. They may house versions locally. They may retain versions in the cloud. Members of your development team may use their own virtual machines or they may be building on a development machine. We find most approaches to be **driven by the specific client environments**.

It's now much less about the tools for dev and QA environments, where people are testing and combining items for the integration side. Instead, it makes sense to adopt shared environments, be it virtual machines somewhere in the **cloud** or on premise in your datacenter. We prefer the hybrid approach in which developers use their own machines and an automated process to send to a combined store or version.

When they push their build out to automate it, they then have access to the latest build from everyone else's store. As each developer issues a release, it folds back into the system and, when requested, the team gets the latest version, which includes everyone else's builds as well. This will also automatically build into testing environments, staging and then production.

Figure 2 shows a full hybrid environment in which the development servers, and user acceptance test (UAT) and production servers are in the cloud. The

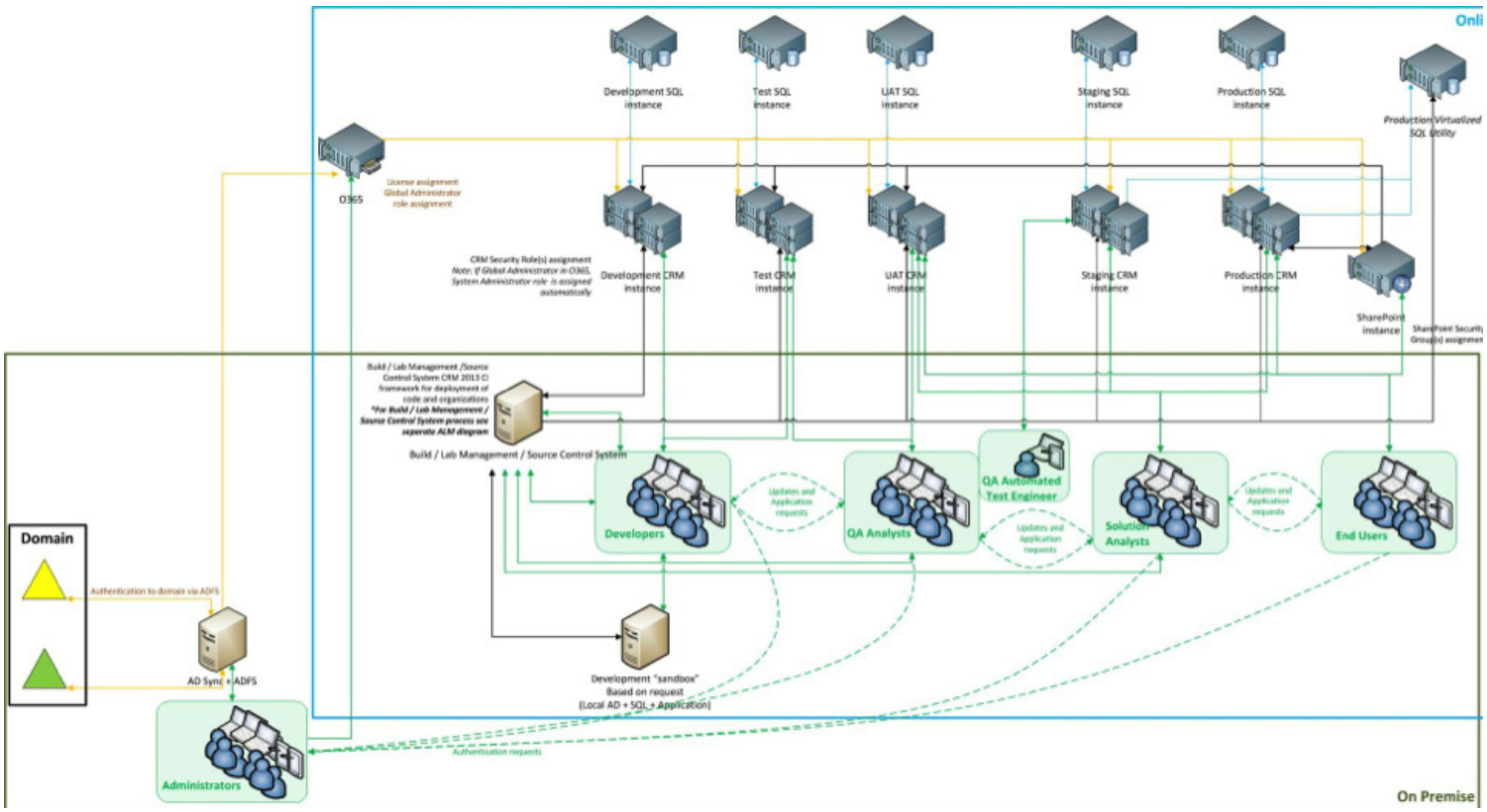


Figure 2

developers can then work on their virtual machines via source control; the build server at the bottom actually gets pushed to the bigger development, merging environments and pushing back to them. This is a fully built out, full lifecycle environment with testers, developers, QA, analysts, even solution analysts. In this case, the solution analysts are experts that go in, test and then find the end users in the environment.

Version Control

With version control, the goal is to rebuild the whole development and production environments from source control. If you can't accomplish that, your version control is not really set to the appropriate level. A lot of people are happy just to be able to verify their source versions, but if you can't fully recover from having to start over again from your version control, you've wasted money. You don't get any of the real benefits of having code versions. Without version control, you're going to endure a bigger workload and longer process to get to your final product.

There are a couple of great developer toolkits that come with the Dynamics CRM 2013 software development kit that allow you to use Visual Studio and talk directly to CRM in order to properly configure. The Solution Package Control consists of a set of customizations and configurations; it will unpack into folders so that you can use source control. Changes made to particular entities are viewed as separate occurrences, providing greater granularity in your source control. This gives you the ability to see precisely what's off or whether something is broken.

A new tool for 2013 allows you to move configuration data. If you have certain requests, like "include all cities or states in this lookup," the tool will enable you to move that information along as you deploy code. The tool unpacks the solution, and then puts it in a folder structure that you can use with your source control system.

Requirements and Testing Strategy

It's imperative that requirements and design tie back to testing. A lot of people focus on the source control and the deployment side of ALM. This is a key strategy in which you can store your requirements and make sure that every bit gets tested. Any bugs are fed back into the system to create a closed loop, ensuring that every requirement becomes a user story that becomes a task; it's essentially a test of the test plan.

A strong requirement, versioning, testing, deployment and bug control system is essential to the delivery of a good product. This is really the only way to ensure quality. Without this quality cycle, you can't accomplish the rest successfully.

There's been a lot of talk around automated testing. Many will tell you, "it's the only way to go," but 95 percent of people who tell you that have never implemented it. The moment you start implementing, you discover it's rather difficult, and sometimes the things that are very obvious to a human tester cost a lot of money to automatically code and test. The amount of time it will take is usually a key price consideration. If you make a change, you can rerun your whole test set for a couple of minutes or a couple of hours, and you're done. You'll know all is as it was before.

But there's always that management decision regarding where to spend your money and what makes sense in terms of putting these initial scripts together, especially if it's UI-based with CRM. In this case, because the UI is so flexible, it's easy to simply drag and drop to make changes. If that's done every time, you have to update your automated testing, which will drive your costs up significantly. Better to focus on unit testing where you can run JavaScript on plugins and then automate integration testing.

This is something you can fully define and mock up, whereas UI testing is easier done by hand and at a function level, not to the nth degree. Figure 3 is an example of a full cycle demonstrating how, if you drill into it a bit, the requirements become a user story, which become tasks.

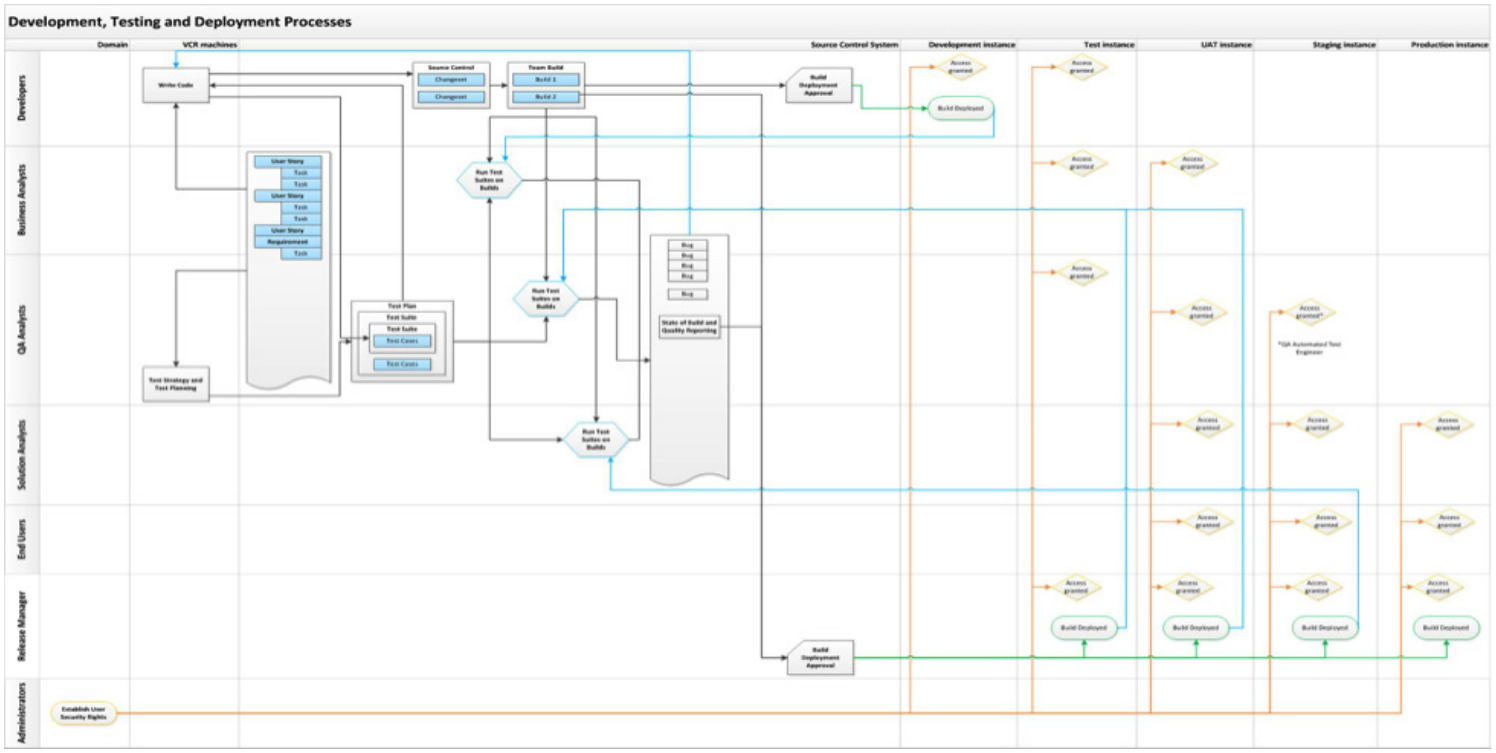


Figure 3

This represents a full, end-to-end enterprise system with governance – the ideal, with all the bells and whistles. This is at the top end; most people tend to achieve maybe 30 to 40 percent of this. But it's always nice to see what a full deployment looks like. It's a good schema to use at the starting point and then say, "What's overkill and what are we prepared to do in the environment?" You should really only be doing all of this if you're a multi-million dollar company. Otherwise, you won't have enough people to complete all sign-off steps.

We use Microsoft's TFS a great deal; it's very popular these days. It allows you to drag current, active tasks as they run through the different stages. There are probably four or five good tools in this space. They generally do the same things, giving you a definable set of requirements to task against. Users also have the ability to drag and drop the tasks they're working on so stakeholders can see the tasks others are working on. They can see what the burn down (resource utilization) is, which tasks are closed, and the capacity others have.

The whole test case aspect of the tools is extremely useful. You can create test cases and define – step-by-step – what the task is. So you've got a manual tester who can see which steps are required. They know what they're supposed to see, and they can take screenshots as they run the test and paste it in to uncover what they didn't see. Someone can actually come back and view the triage to gauge if it's a bug and put it back into the task list to fix.

Deployment Strategy

A deployment strategy can encompass test environments for functionality or UAT for performance. Each has its place but UAT is more akin to your production environment. You can actually test that space – stress testing and performance testing.

'Mock' is a functionality that is fairly self descriptive. If you're writing a certain function of a particular component, you can actually mock losses around it. You can then look at the result and put in what are called asserts to make sure you get the proper reaction. This method is used in conjunction with unit tests.

It takes a bit more work to write the full mock of what your integration is going to look like, but if you write the mock strictly to the specifications of your interface, you can do a lot of pretesting before real integration. I recommend mocking because it decouples the timelines a bit; otherwise you've got a case of the chicken and the egg. Both sides are writing interfaces to a central point. One side has to wait for the other side to start testing. It's better to have a mock in the middle. You can test against it until you're ready to perform the integration.

Governance plays a key role in your deployment strategy. It ensures that all external and internal parties have a clear picture of deployment and how it affects stakeholders outside the project.

There are a couple of excellent tools that can greatly benefit your deployment strategy. ALM Tool Build Manager in Visual Studio Online enables more sophisticated development and testing. The xRM CI Framework is a set of tools that allows you to quickly and easily implement Continuous Integration for your Dynamics CRM solutions. The Adxstudio ALM Toolkit is a suite of tools that helps automate change management for Microsoft Dynamics CRM projects using a source control system such as Microsoft Team Foundation Server.

Data Strategy

When determining your data strategy, you can use configuration data, test data or data obfuscation. Configuration data is key to automation, as well in ensuring systems are configured properly. There are times when you want to recreate a new test for an issue that you really can only conduct with data that's representative of live data rather than created test data. There are some tools that are good at obfuscation – they obscure your production data by changing names, phone numbers, addresses, any recognizable data that shouldn't be readily revealed.

Existing tools are great and you can write your own as well. If you take snapshots in your testing, you've got to place the right security around this data. Even in a test environment, production data must be secured.

Determine Your ALM Approach

There is no one, right way to tackle ALM. Instead there are a variety of methods that should be evaluated, selected and blended for your unique scenario. You should pick and choose what makes sense for your organization to come up with the best approach.

Bottom line, your requirements should drive your ALM process. Throughout it, you'll want to measure how closely you met your set objectives and how many issues have arisen. It's all about tracking. Have best practices in place as each project starts. You can certainly build on it later, and a lot of people do. But it's going to be twice as hard to build it later than it was to start from the beginning. What would have taken one day becomes a week when you're already halfway through.

Focus on testing and testability. And note that some things are worth automating. If your environment allows it, build continuously and test constantly. This will empower you to push applications to market quicker. Why wait for a massive release cycle if you can keep on pushing and making your customer's happy.

Know that you **don't** have to do everything from scratch. There are great tools available to create a simpler route. Don't over-complicate the process. Do what works continuously and learn using your knowledge and history to make it better, not more complex.

Contrary to public opinion, ALM processes can be fully implemented in a Dynamics CRM environment. Sure, there are a lot of people who say, "Oh, it's too hard," but it's actually pretty easy. Just follow the example of what other people have done to get it done. The end result will be better quality applications.

About the Author

Riaan van der Merwe is a general manager for Dynamics at Neudesic, a Microsoft National Systems Integrator and Gold Certified Partner, dedicated to the Dynamics CRM platform. Van Der Merwe brings 25+ years of experience in architecture strategy and application development management. Prior to Neudesic, he was a global architect and development manager at Jones Lang LaSalle, where he led the strategy and deployment of the firm's intranet, internet, extranet and application deployment frameworks, as well as the firm's CRM and SharePoint strategies, Before Jones Lang LaSalle, he consulted financial services and real estate companies.