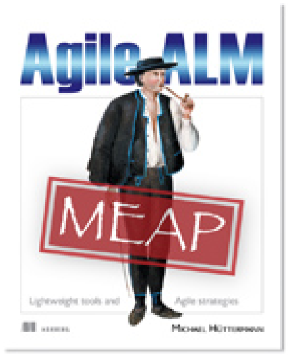
MANNING PUBLICATIONS



Agile ALM
By Michael Hüttermann

There is a conflict between having too much lifecycle management process and not having enough. An Agile approach advances and demands feedback and communication. Application Lifecycle Management (ALM) and its major facet release management have to be a balanced set of process and tools aligned to your individual requirements. In this article from Agile ALM, you will learn about the facets of Agile ALM and how it plays the role of a change enabler.

You may also be interested in…

Agile in the Context of Application Lifecycle Management (ALM)

There is a conflict between having too much lifecycle management process and not having enough. An Agile approach advances and demands feedback and communication. The release management process should be effective, efficient and targeted. In practice though, many projects suffer from not having enough process. You need to focus on priorities including the root cause of the issue. Often, more process is introduced in order to address the problem. Introducing too many rules or wrong process rules could suggest control that does not really exist. In the worst case, you have a *described process* and *a real process* in parallel. Or you have a very rigid process that decreases productivity dramatically. Application Lifecycle Management (ALM) and its major facet release management have to be a balanced set of process and tools aligned to your individual requirements.

Effectiveness and efficiency

Simply speaking, effectiveness is doing the right thing, and efficiency is doing the right thing right. After consulting on a significant number of projects, I'm left wondering why some teams do not grasp this distinction. If you have issues (or, better, challenges), try a root cause analysis to detect the original evil. If you find it, you can think about possible improvements. Mostly, they all have pros and cons, so you should decide wisely which way to go. You should only go a few new ways in a trackable amount of time, to measure the success of your decisions. If you dig into challenges deep enough, you most often find communication defects inside the team. That is what Agile is all about. Communication and interaction is more important than processes and tools, as the Agile Manifesto says. If you can solve the people issues yet still see room for improvement, proceed to the processes.

Defects in processes are often a problem. For example, it is not possible to configure a workflow system to cover your processes, unless you know the processes. If they are not described, identify and describe them. Sometimes, processes don’t exist at all. Set them up; don't be satisfied if the whole team just speaks about the task of “daily business”. If you are managing the processes, and you know the requirements, then, and only then, can you think about tooling. A bad example would be to buy a full-fledged commercial ALM suite when you do not know your requirements (and, therefore, are not able to validate if the tools do fulfill them).

You can work with prototypes, evaluation versions of tools, or a release 0.0/zero for setting up infrastructure. These provide good possibilities to get early feedback and gain some valuable experience. But, always remember that you should stay flexible. It is often better to use a collection of lightweight, integrated tools that are de facto standards on the market and that do the best job in their domains. You can integrate and decouple your infrastructure, still remaining quite independent and flexible.

If you want to kick-start your development of new components, you may decide to use Maven, which provides component and build management and a neat archetype feature. If you want to integrate your system continuously, you can add a build server to your infrastructure. You may want to add tests and audits later. Little by little, you can extend your infrastructure in a requirement-based, focused way. And, if you are not satisfied with one decision, you can replace one tool while still sticking with the other ones.

Another core competency is managing the identification of configuration items.

Agile ALM and configuration items

ALM deals with management of tasks and artifacts. Controlling artifacts is only possible if the artifacts are identified: without deciding which artifacts have impact on the release and the project and without putting the artifacts into the ALM system it is not possible to carry out controlling, status accounting (validation of completeness to provide a consistence version), and audits. Additionally, setting up an efficient ALM is only possible when processes and tools are optimally chosen, integrated and standardized.

Identification of assets, controlling, status accounting, and audits are major tasks of traditional Software Configuration Management (SCM). In an Agile ALM, you'll find the best fit to implement SCM. There should be an SCM-aware expert on every Agile team. Pure Agile projects do implement SCM facets in an explicit way. Driving the SCM needs of the business daily can be done by the traditional build manager or a (technical) release manager. This depends on how you slice your roles.

SCM is mainly about access to project artifacts. This includes not only tracking artifact versions over time but also controlling and managing changes to them. Whereas in traditional SCM scenarios you track every single artifact, in an Agile ALM scope you will focus on final deployment units and important artifacts including documents influencing the project (like requirement documents). As an example, the Agile approach tracks Enterprise Archive files (EARs), Web Archive files (WARs), and Java Archive files (JARs) regardless of their contents (packages, classes). The sources are stored in the version control system. You will not store artifacts that you can generate out of other artifacts (unless you have good reasons for that) and documents that will not change over time or by multiple users (for example, meeting minutes), as seen in figure 1.

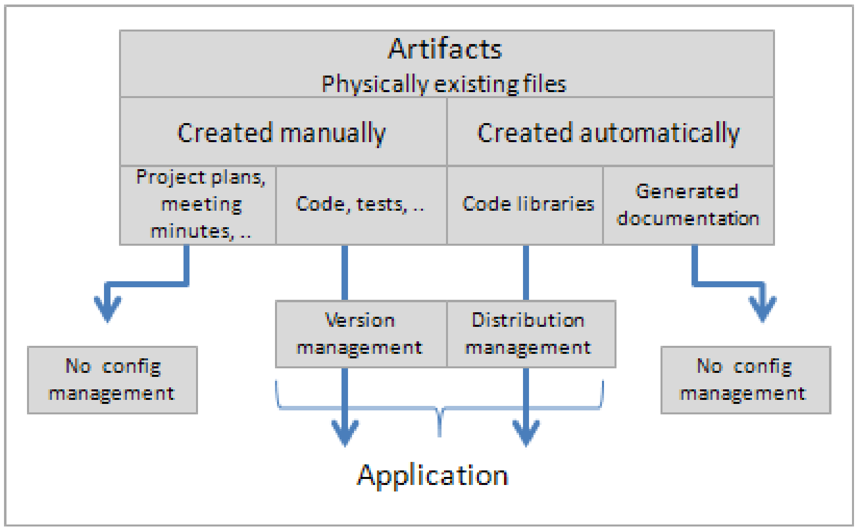


Figure 1 Artifacts in configuration management. Artifacts that are updated continuously and are of special interest are put into configuration management. Sources and documents are put into version control; libraries are put into distribution management.

From an underlying SCM point of view, an Agile ALM focuses on aggregating and documenting the most important parts necessary for releasing. For example, if you want to integrate further components or subsystems into your enterprise SCM, you need basic information about these components, including their deployment units. Table 1 collects some of the major ones in an SCM checklist. This is a much leaner approach than promoted by traditional

SCM. Depending on your individual situation and requirements, implementing the checklist can be done in a smart way. If you use Maven, for instance, some of the checklist items are covered out of the box (like documenting deployment units). Tools like Maven help to define your SCM in an executable medium. This means, you do not only document on paper—you have XML definitions that can be executed reproducibly.

Table 1 Approaching SCM in an Agile way: the SCM checklist

Group	**Item**	**Details**
Overview	Configuration elements	A complete list of all configuration elements, including scripts, DB elements, deployment units, properties
System	Deployment diagram	Deployment units (and their versions), packaging types, protocols, technical information (like WAS 6.1), dependencies between configuration elements, nodes
System	Infrastructure	Database elements (users, DDL), technical users, permissions, security
System	Test environments	For all subsystems, mapping to other test environments where needed
Build	Build system	The system must provide its deployment units in a reproducible way (build must be provided by component development team!)

Traditional SCM requires listings of configuration items (similar to the expression *work items*) and checklists. This can be necessary in an Agile context too but is usually handled automatically in ALM tools. But, there are many possible usage models. Release notes should be created automatically. You should also be able to conduct a system configuration audit automatically. For example, JIRA can be used to derive release notes based on a specific time interval or version number. Mapping sources to requirements to generate what you're looking for is an effective way to create documentation automatically and apply an active Impact Management, with Mylyn or FishEye. Acceptance tests (for example, with Fit) can also be part of this documentation.

Tests and audits (metrics) can be applied automatically as part of the continuous integration system. Here, you can base your audits on Checkstyle and Cobertura and your testing on, for example, Selenium, FEST, or something similar. Track your components in a component repository, where the system puts them continuously. If you already use Maven, you are familiar with this.

Agile ALM minimizes overhead while maximizing benefit. It also acts as an enabler for change.

Agile ALM as change enabler

All systems try to achieve stable states (Panta Rei[1]). Being flexible and Agile in software development does not mean chaotic drifting but being able to change and transform from one stable state to another.

The importance of lifecycle management will continue to grow. In a time of distributed, heterogeneous system landscapes, integration of legacy systems and components in many different versions and (transitive) dependencies on different platforms, following a systematic release management approach has become a

[1] Meaning "everything flows"

precondition to providing software of high quality in constant, short intervals. Agile ALM is the catalyst enabling the daily work of all project stakeholders. It also helps to track and control the artifacts that were created during the project activities.

During the development of complex systems, change is a constant companion of the development process. Instead of being exceptional, changes are becoming more and more the norm. A very high percentage of projects miss their project goals because they did not grant enough space for changes in the process. Modern software development understands changes to be a major part of the project. They are part of the process in order to align the current activities to the valid requirements and basic conditions at any time (continuous adaptation).

In the extreme approach, defects (bugs) and all kinds of functional and non-functional requirements are handled like a coordinated set of changes to the system. Following this paradigm, software development is the process of identifying and processing changes. ALM is evolving to be the hub of reproducibility and a change enabler.

Summary

We talked about what Agile means in the context of ALM. We discussed some aspects of the Agile ALM, including using SCM checklists and being both effective and efficient. Finally, we considered Agile ALM as a change enabler.

For source code, sample chapters, the Online Author Forum, and other resources, go to
http://www.manning.com/huettermann/

Here are some other Manning titles you might be interested in:

Becoming Agile
...in an imperfect world
Greg Smith and Ahmed Sidky

Specification by Example
Gojko Adzic

Enterprise Search in Action
Marc Teutelink

Last updated: July 26, 2011