Practical Application Life cycle Management (ALM)

2009-02-18
조대협
(http://bcho.tistory.com)

Table of contents

Overview

ALM의 정의를 wikipedia에서 찾아보면 다음과 같다.

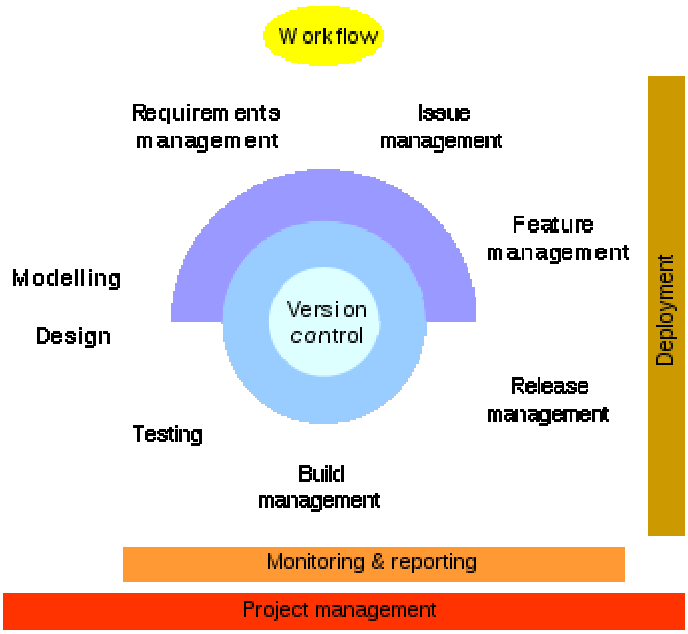
> Application lifecycle management (ALM) is the marriage of business management to software engineering made possible by tools that facilitate and integrate requirements management, architecture, coding, testing, tracking, and release management.[1]

해석해 보자면, 기존의 애플리케이션 개발은 기술적인 관점에서 많이 접근이 되어 왔으나, 비즈니스 요건 관리 부분과 실제 소프트웨어 개발 프로세스를 융합하고 이에 대한 관리를 자동화된 툴을 이용하도록 하는 것이 ALM의 개념이며, 이 부분에는 요구 사항 관리, 아키텍쳐, 코딩, 테스팅, 그리고 이슈 추적과 릴리즈 관리등을 포함한다.

한마디로 이야기 하면 비즈니스와 실제 소프트웨어 개발간의 괴리를 없애고, 소프트웨어 개발의 요구사항 분석에서부터 릴리즈까지의 모든 과정을 툴을 도입함으로써 관리하겠다는 개념이다.

기존에 소프트웨어 개발 프로세스는 요구 사항 관리에 대한 문서나 시스템, 아키텍쳐나 디자인에 대한 Case tool과 산출물, 코드 관리, 일정 관리 등등이 각기 다른 제품과 다른 프로세스 다른 템플릿으로 구현이 되어 왔고 이로 인해서 소프트웨어 개발 과정에 대한 개념이 실제로 구현되었을때는 단계별로 추적성과 실용성이 떨어졌다.
ALM의 의미는 이런 현실과 괴리된 부분을 좀더 통합되고 현실적으로 전문화된 도구를 이용하여 현실화 시키고 궁극적으로 소프트웨어 개발 프로세스를 개선하는데 목적을 두고 있다고 말할 수 있다.

ALM이 실제 커버하는 범위를 보면 아래 그림과 같이 요구사항 관리에서부터 프로젝트 관리, 릴리즈 관리까지 소프트웨어 개발의 거의 전 영역을 커버하는 것을 볼 수 있다.



(위키피디아 ALM Concepnt 그림 인용 : http://en.wikipedia.org/wiki/Application_lifecycle_management)

요즘 들어서 많은 업체들이 이 ALM이라는 개념을 사용하고 있다. H*의 경우 버그 추적 시스템을 기반으로 이슈 관리, 테스트 자동화 도구를 가지고 ALM이라고 이야기 하고 있으며 Mc* 라는 업체의 경우 테스팅 툴 하나만을 가지고 ALM 업체라고 이야기 하고 있다.

ALM이 가져야 할 최소한의 요건은
요구 사항 관리, 프로젝트 스케쥴 관리를 위한 Task 관리, 빌드 환경 자동화 및 형상 관리, 테스트 자동화 부분이 핵심이라고 본다.
그외에 Deployment의 경우 주로 운영 (SM : System Management) 조직이 담당하며, Design이나 아키텍쳐링의 경우 ALM 사상내에 포함되는 것이 이론적으로는 맞지만, 사실 Design의 경우 프로세스를 정형화 시키기 어려울뿐더러, 실제 프로젝트에서는 Design이 프로젝트가 진행되어감에 따라 변화하고 완성되어 가기 때문에, Design을 포함시키는 것은 쉽지 않다고 생각한다.

사람들은 프로젝트를 하면서 똑똑해지고 시스템은 프로젝트 진행됨에 따라 명확해 진다. Agile 사상에서도 Design은 선행 작업이 아니라, 주로 프로젝트 진행과 같이 가는 On going 작업으로 정의하고 있다. Design을 ALM의 구현체 내에 포함시키기 위해서는 말단 개발자까지 상당 수준의 성숙도가 필요하다.

문제점

ALM의 사상적인 출발은 툴을 이용한 소프트웨어 개발 사이클의 현실화인데, 시장에 있는 툴의 경우 그 성숙도가 매우 높아서 프로젝트에 적용하는데 상당한 경험과 지식을 필요로 한다. 실용적인면에서 생각했을 때 ALM의 적용 범위는 일반 말단 개발자 수준에 까지 적용이 되어야 하기 때문에 **난이도**가 높을때는 실제 프로젝트에 적용하기가 어려운 점이 많다. 또한 ALM Company를 자칭 하는 많은 회사들이 ALM에 대한 Full set을 가지고 있지 않은 상태에서 **마케팅적인 메시지로** Drive 하는 경우가 많아서 사용자의 혼란을 초래하고 있다.

그리고 무엇보다 중요한 것은 ALM을 시스템으로 구축하기 위한 제품이 아니라 ALM을 조직에 적용하기 위한 프로세스와 방법론 즉, 컨설팅과 같은 인적 지원면인데, 적어도 국내의 벤더들에서는 실용적으로 ALM을 프로젝트에 적용할 수 있는 업체가 있는지는 의문이다.

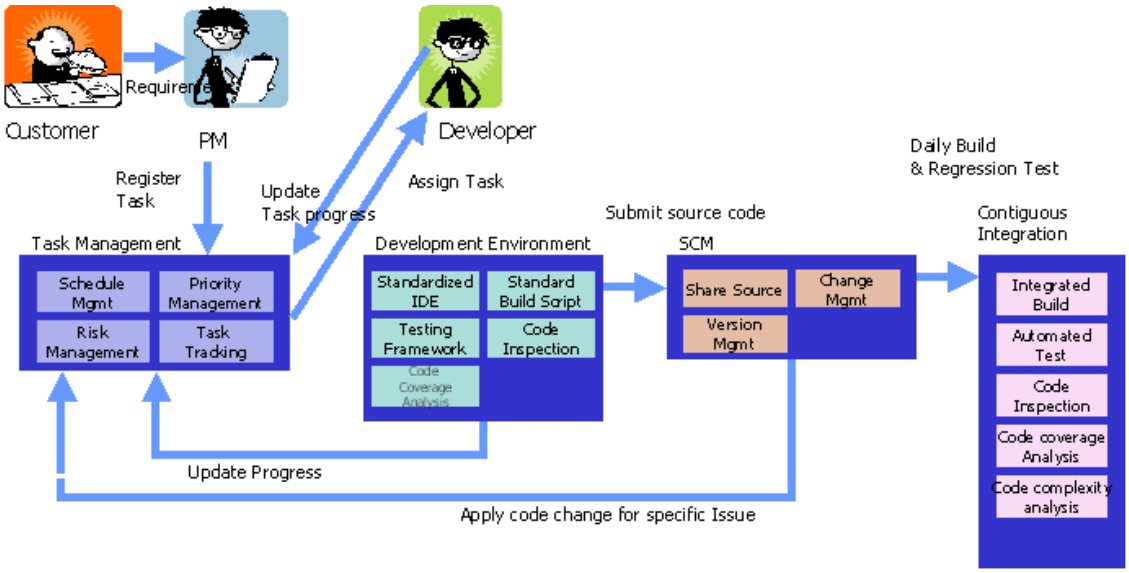
실용주의 ALM

실용주의 ALM은 이런 문제점을 바탕으로 좀더 실용적이고 실무적인 ALM을 개발하여 실무에 사용하고자 하는데 목적을 두고 있으며 아래와 같은 특징을 갖는다.

- 주로 오픈소스나 저비용의 제품을 조합하여 ALM의 핵심 범위를 커버한다.
- Agile과 Kent Beck, Erich Gamma, Joel Spolsky 등에 의해서 주장되고 있는 실용주의 방법론 (Practical methodology)를 바탕으로 하여, 튼튼한 이론적인 바탕을 가지고 현실에 맞는 실용적인 프로세스를 구축한다.
- 구현팀을 위주로 프로세스를 정의한다.
- 품질 향상

End to end use case (시나리오)

이해를 돕고자 이제부터 소개하고자 하는 실용주의 ALM 프레임웍의 가상 시나리오를 살펴보도록 하자



개발자 시나리오

1) 개발자 D씨는 아침에 출근해서 이클립스 IDE를 오픈한다.
2) 이클립스 IDE는 이슈추적툴로 부터 D씨에게 할당된 작업들이 리스트업되고, 그중에서 오늘해야 할 작업을 선택하여 PROGRESS로 상태를 변경한다.
3) SCM으로 부터 최신 코드를 UPDATE받고 코딩을 시작한다.
4) 코드를 만들고 테스트 케이스를 작성하여 코드가 제대로 작동함을 확인하고, 커버러지 분석을 통해서 금일 코딩한 내용이 테스트에서 모두 확인 되었는지 체크한다.
5) 오늘 코딩한 내용을 INSPECTION툴을 통해서 잠재적인 문제가 있는지 없는지 검증받고 NAMING RULE등이 문제 없는지 확인한다.
6) 완료된 내용을 SCM에 작업 번호와 함께 COMMIT한다.
7) 자동 빌드 머신에서 COMMIT된 소스코드를 감지하고 모든 소스를 내려받아서 빌드를 완료한후에 개발 서버에 자동으로 배포하고 테스트를 수행한다.
8) 테스트가 실패한 경우 이전 버전으로 개발 서버를 원복 시키고 모든 개발원과 PM에게 이메일로 테스트 실패 사실을 통보한다.
9) PM은 테스트 실패사실을 이메일로 통지 받고, 이번 빌드에서 변경된 부분을 빌드 자동화 시스템을 통해서 확인하고 빌드 자동화 시스템에 의해서 리포트된 내용에 따라 누가 어느 모듈을 수정했는지를 찾아서 해당 개발자에게 수정을 지시한다.
10) 개발자는 수정을 마친후에 다시 SCM에 소스를 반영하고 빌드 자동화 시스템은 빌드,테스트,커버러지 분석,코드 복잡도 분석,INSPECTION작업을 수행한다.
11) PM은 빌드가 완료된 결과를 통보 받고, 복잡도가 높은 클래스 모듈 10개에 대해서 제대로 테스트가 커버하는지 확인하후에 미비한 부분에 대해서 담당 개발자에게 테스트 보강을 지시한다.

PM의 작업지시 시나리오

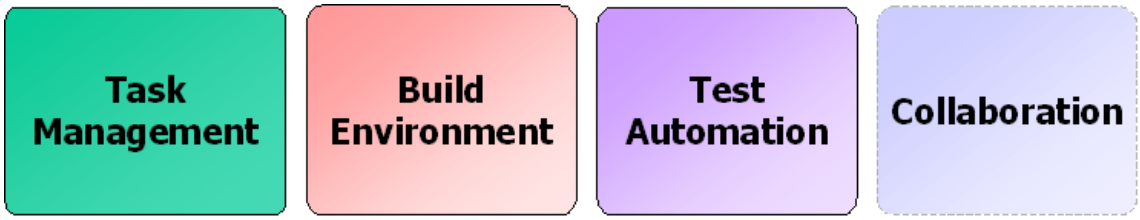
1) PM P씨는 아침에 출근하여 고객으로 부터 새로운 요건을 받았다.
2) P씨는 요건을 정리하여 내용을 이슈 트랙킹 시스템에 등록하고 심각도와 우선 순위를 지정해서 개발 PL에게 ASSIGN하였다.
3) 개발 PL은 요건의 우선 순위와 긴급도를 보고 누구에게 작업을 지정할것인지 고민한다. 이슈 트랙킹 시스템에서 개발원별로 진행중인 이슈 사항을 보고 개발 난이도에 맞는 사람중에서 가장 할당된 이슈가 적은 사람에게 이슈를 ASSIGN하였다.
4) 개발자는 해당 ISSUE를 받아서 처리한 후 PL에게 다시 ASSIGN한다.
5) 이때 작업에 관련된 메일,전화 통화내용 기타 관련 내용들을 시스템에 모두 LOGGING한다.
6) PL은 작업이 완료된 내용을 검토한후 해당 ISSUE를 CLOSE한다.
7) PM은 자신이 지시한 내용에 대해서 누가 진행하고 있으며 진행현황이 어떻게 되었는지를 이슈를 통해서 추적할 수 있다.

PM의 현황 관리 시나리오

1) PM P씨는 1차 오픈 때까지 해결되어야 할 이슈를 이슈 관리 시스템에서 검색한다.
2) 심각도가 높은 이슈와 오픈된지 오래된 이슈를 확인하여 진행이 안되고 있는 이유는 무엇인지 RISK는 무엇인지를 조사하고, RISK에 대한 대비책을 세운다.
3) 만약에 날짜에 비해서 해결되어야 할 이슈가 많을 경우 심각도와 우선순위를 고려하여 2차 오픈으로 이슈를 연기한다.

실용주의 ALM 구성

실용주의 ALM은 크게 4가지 모듈로 구성된다.



Task Management (Task 기반의 프로젝트 관리)

프로젝트에서 진행되는 작업에 대한 진행 상황과 스케쥴링 그리고 리소스에 대한 관리 방법론을 제공한다. 기본적인 사상은 Agile 방법론의 프로젝트 관리 방안을 기반으로 하고 있다. Task Management 모듈은 다음과 같은 서브 모듈을 포함하고 있다.

Process

1) **Task Management Process** : Agile Scrum 기반의 Task 관리 프로세스
2) **SOA or Open API Service Lifecycle Management Process (Optional)** : SOA나 WEB 2.0에서 Service Lifecycle (서비스 신청에서부터 배포 까지) 관리 프로세스와 시스템
3) **Requirement Analysis Process (TBD)** : 요구 사항 추출 프로세스

Reference Architecture

1) **Task Management System Reference Architecture** : 프로세스를 구현한 Task 관리 시스템
2) **Task Management Dash board Reference Architecture** : 프로젝트의 진행 상태를 한눈에 알 수 있는 Dash board
3) **Requirement Analysis Management Reference Architecture** (TBD) 요구 사항 관리 시스템

Build Environment (빌드 환경)

Implementation 단계에서 사용되는 개발환경을 제공한다. 기본적인 사상은 Pragmatic(실용주의) 방법론으로 Kent beck이나 Joel Spolsky에 의해서 소개된 방법론을 기초로 하며, 일일 빌드와 점진적/반복적 통합론을 중심으로 개발환경 구성에 대한 가이드를 제공한다.
다음과 같은 서브 모듈을 포함하고 있다.

Process

1) **SCM branch guide** : 소스 형상 관리에 대한 브렌치 관리 전략
소스 코드 관리에 있어서 브렌치에 대한 관리 방법을 기술한다. 브렌치는 잘못쓰면 전체 소스코드 관리를 하는데 있어서 엄청난 혼란을 초래할 수 있는 만큼 적절한 브렌치 관리 정책이 정해져야 한다.

2) **Contiguous Integration Process** : 점진적 통합 방법에 대한 프로세스
자동 빌드 시스템을 구축하여, 개발자의 소스 코드 변화를 자동으로 인지하고 매일 자동 빌드를 통해서 코드의 통합을 빅뱅 방식이 아니라 매일 점진적인 방식으로 진행함으로써 통합으로 인해 발생할 수 있는 문제를 조기에 발견하고 해결할 수 있도록 하며, 빌드 과정내에 테스팅을 포함시켜서 결함을 조기에 발견하여 전체 소프트웨어 품질 향상을 유도 한다.

Reference Architecture

1) **Standard IDE** : 개발자를 위한 표준 개발 환경 가이드
프로젝트에서 개발자별로 선호하는 개발 도구들이 다르다. (이클립스, NetBeans, VI 등) 다른 개발 도구는 구현된 코드에도 영향을 주며, 특히 새로운 팀원이 합류했을 때 보통

1~2일을 개발환경을 셋업 하는데 시간을 소요 하게 된다. 표준 개발 환경은 ZIP으로 개발에 필요한 모든 환경을 만들어서 개발자가 ZIP 파일만 풀면, IDE에서 테스트 프레임웍, 테스트용 서버 환경까지 일괄로 셋업이 되서 모든 개발자가 빠른 시간내에 동일한 개발 환경에서 소프트웨어를 구현할 수 있도록 한다.

2) Contiguous Integration Reference Architecture

실제 CI 환경을 구축하는데 필요한 아키텍쳐에 대해서 설명한다.

3) **Standard Build Script** : 표준 빌드 스크립트 구축 가이드

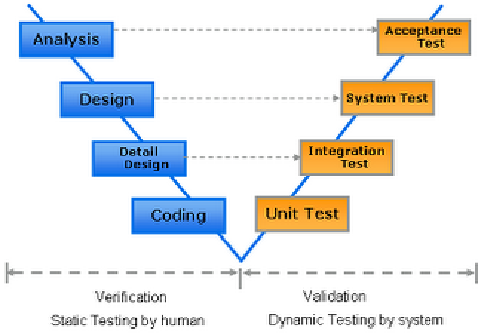
표준 개발환경과 마찬가지로 빌드 스크립트 역시 표준화 되어야 한다. LIB 위치나 버전, 빌드 스크립트내의 TARGET 정의, 빌드 순서등을 통합하고 표준화된 형태로 제공하여 개발자가 빌드 스크립트 작성에 소요되는 시간을 절약하고, 비표준화된 빌드스크립트 사용에서 오는 오류를 예방한다.

Test Automation (테스트 자동화)

일반적으로 테스팅은 QA팀의 역할로 인식이 되어 왔고, 시스템 개발이 완료된 후에 QA팀에 의해서 테스팅이 되는 것이 일반적이었다.

여기서는 소프트웨어 개발중에 개발팀에 의해서 수행되는 단위 테스트와 통합테스트에도 비중을 두고, 테스트 자동화와 회귀 테스트(테스트 마다 지난번 테스트했던 내용을 포함하여 테스트 하여, 변경 사항이 기존 기능에 영향을 줬는지 여부를 검증함)를 중점적으로 다룬다.

테스팅 모델은 전통적은 Waterfall 방식을 확장한 V-Model을 기반으로 한다.



Process

1) **Testing Process** (테스트 프로세스)

소프트웨어 개발 단계에서 부터의 단계별 테스팅 프로세스에 대해서 정의한다.

V-Model에 기초하여 Unit Test,Integration Test, System Test, User Acceptance Test 4단계로 나누어서 정의한다.

2) **Defect Management Process** (결함 관리 프로세스)

테스트 결과 발견되는 결함에 대한 관리 프로세스를 정의한다.

Reference Architecture

1) Unit Test Framework Reference Architecture

특히 단위 테스트의 경우 소프트웨어의 안쪽을 테스트 하기 때문에, 일반적인 테스팅 도구보다는 소프트웨어 컴포넌트에 따라 더 정밀한 테스팅 도구가 필요하다.

단위 테스트를 수행하는데 필요한 테스팅 프레임웍에 대해 정리한다.

2) Static Testing Reference Architecture

소프트웨어 테스팅 기법중에서, 소프트웨어의 동작 상태가 아니라 코드 검증을 통해서 결함을 찾아내는 방법을 Static Test라고 한다. 이 테스트에서는 코드의 Naming Convention이나 특정 패턴에 따른 잠재적인 결함 (메모리 누수, Null Pointer Exception)을 찾아낼 수 있다. 이 Static Test를 수행할 수 있는 도구에 대해서 설명한다.

3) System Test Reference Architecture

테스팅 중에서 주로 성능과 비기능 (확장성, 가용성등)에 대한 테스팅 수행 도구에 대해서 설명한다.

Collaboration (협업)

협업 모듈은 팀이 프로젝트를 진행하는데 있어서 의사 소통과 공동 작업을 돕기 위한 몇가지 기법과 시스템에 대해서 소개한다.

Process

1) Code Review

코드 리뷰는 실제로 코드를 검토하여 예측 되는 결함을 찾아내고 서로 개선 방향을 찾아내는 행위이다. 코드 리뷰의 방식은 여러가지가 있으나 비형식적인 코드리뷰라도 투자대비 소프트웨어 품질에 많은 효과를 줄 수 있기 때문에, 협업의 기법중의 하나로 소개한다.

Reference Architecture

1) Wiki Based Document Management Reference Architecture

ALM을 이용해 구축된 표준이나 프레임웍, 프로세스등 많은 내용들이 Architect 레벨에서 일반 개발자들에게 전달되어야 한다. 이런 지식을 전달하는 방법이 문서등 여러가지 방법이 있겠지만, 문서등은 여러 버전 관리나 변경 관리가 어렵고, 표현의 한계가 있다.

Wiki의 경우 내용을 하나의 장소에서 계속 업데이트가 가능하고, 검색이 가능하며, TEXT와 이미지뿐만 아니라 멀티미디어 데이터를 넣을 수 있기 때문에 직관적인 정보 전달이 가능하다.

또한 링크를 이용하여 정보간의 연관 관계를 정의할 수 있다.

MS-WORD등으로 만들어진 문서는 공유 폴더에서 사장되거나 또는 이쁘게 바인딩되어

서 책장이라는 무덤속으로 사라질 수 있는데, Wiki를 이용하는 이유중의 하나는 꼭 필요한 문서만, 필요할때 사용할 수 있도록 하여, 프로젝트에서의 정보 공유를 가속화 하는데 그 목적이 있다.

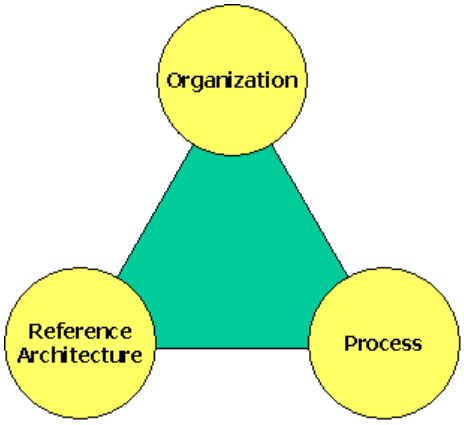
2) Communication with Forum Reference Architecture

Wiki의 경우 대부분 위의 조직에서 아래 조직으로의 하향성을 가진 단방향 커뮤니케이션이다. 이를 보안하기 위해서 Forum은 좋은 양방향 커뮤니케이션 도구로 사용될 수 있는데, Wiki를 통한 단방향 정보 전달의 Feedback으로 사용할 수 있다.

이러한 협업 도구들은 전체 협업의 생산성을 높혀줄 수 있는 일부만을 권장하는 것이며 조직의 수준과 구조에 맞춰서 별도의 협업 프레임웍을 구축 하기를 권장한다

실용주의 ALM 의 구현

ALM을 프로젝트에 적용 하기 위해서는 아래와 같이 3가지 관점에서의 접근이 필요하다.



프로세스

ALM은 전체 소프트웨어 개발 프로세스를 커버하기 때문에, 각 모듈을 프로젝트에 적용되는 프로세스에 대한 정의가 필요하다.

Reference Architecture

프로세스를 실제로 시스템으로 구현하기 위해서, 어떤 형태의 시스템 아키텍쳐를 이용하여 프로세스를 현실화할 수 있는지에 대한 가이드를 제공한다.

조직

시스템과 프로세스를 가지고 프로젝트에 적용할때, 적용하는 주체의 역할과 책임에 대해서 정의한다.

성공적인 ALM 구현 전략

간략하게 실용주의 ALM에 대해서 살펴보았다. 이 실용주의 ALM을 성공적으로 적용하기 위

해서 몇 가지 전략이 필요한데, 다음과 같다.

1) Liquid

전체 프로세스가 모난 부분이 없이 물 흐르듯이 하나의 프로세스로 연결되어야 한다. 프로세스가 넘어가는 단계가 매끄럽지 못하면 전체 개발 프로세스에 병목이 생기게 되고, 프로세스 흐름에 문제가 생긴다.

2) Seamless

앞에서 소개한 실용주의 ALM의 모듈 구성을 보면 알겠지만, 상당히 많은 부분을 많은 기술로 커버하고 있다. 이러한 기술들이 유기적으로 결합되어 마치 하나의 통일된 프레임웍과 같은 형태를 취하여 사용자 입장에서 하나의 프레임웍을 쓰는 듯한 느낌을 줘서, 사용자 관점에서 올 수 있는 혼돈을 미연에 방지해야 한다.

3) Process Oriented

ALM의 가장 중요한 요소는 프로세스이다. ALM은 소프트웨어 개발 프로세스를 시스템화 하는 것이기 때문에, 프로세스 자체가 중요하며 자칫 잘못하면 시스템 구현에 이끌려서 프로세스가 망가지는 경우가 있다.

4) Step by Step

실용주의 ALM이 다른 ALM에 비해서 경량이고 현실적이라고는 하지만, 커버하는 영역이 상당히 넓다. 한번에 전체 개발 프로세스를 변경하는 것은 구성원들에게 큰 혼란을 초래할 수 있기 때문에, 난이도별로 단계적으로 적용하는 것을 권장한다.

5) 팀의 수준에 맞춰서

팀의 성숙도에 맞춰서 실용주의 ALM을 Customization해서 적용해야 한다. 성숙도가 낮은 팀에 실용주의 ALM을 적용할 경우, 마치 기존의 중량의 방법론을 적용할때와 마찬가지로 형식 지키기에만 급급해지고 실제 생산성은 오히려 더 떨어질 수 도 있다.

6) Be Simple

모든 기능을 커버하려 하지 말고, 목표가 100일때 80만 커버하더라도 단순성을 우선시해야 한다. 복잡도가 높아질 수 록 실용주의 ALM 사용으로의 진입장벽과 Learning Curve가 급격하게 올라가고 이는 또 다른 형식적인 방법론으로 전락할 수 있다.