|  |  |
| --- | --- |
| **전사 EA 표준 Profile정의서** | |
| **사내 표준 정의 사업**  요구사항 변경관리 툴  **Version 1.0**  Architect/SWAT | |
|  |  |
| SK주식회사 | |

**제.개정 이력**

|  |  |  |  |
| --- | --- | --- | --- |
| **버전** | **변경일자** | **제.개정 내용** | **작성자** |
| *1.0* | **2025-04-14** | *최초 등록* | *장우승* |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Index / 목차

[1. 아키텍처 4](#_Toc196227006)

[1.1. 도입 목적 4](#_Toc196227007)

[1.2. 도입 전 선결과제 5](#_Toc196227008)

[1.3. 핵심 아키텍처 기술 요소 6](#_Toc196227009)

[1.4. 핵심 특징 및 장단점 비교 7](#_Toc196227010)

[2. 시장 상황 9](#_Toc196227011)

[2.1. 시장 동향 및 기술 연계 영향도 9](#_Toc196227012)

[2.1.1. 시장 상황 및 동향 9](#_Toc196227013)

[2.1.2. 기술동향 영향도 평가 10](#_Toc196227014)

[2.2. 시장 점유율 및 주요 기업 채택 사유 10](#_Toc196227015)

[2.3. 주요 요구사항 변경관리 툴 완성도 평가 11](#_Toc196227016)

[2.3.1. 도구별 완성도 평가표 12](#_Toc196227017)

[2.3.2. 평가 항목별 점수 기준표 12](#_Toc196227018)

[2.4. 기술 트렌드에 따른 영향도 및 이유 13](#_Toc196227019)

[3. 기술 동향 15](#_Toc196227020)

[3.1. 전통적인 요구사항 변경관리 툴(도구)들의 한계점과 시사점 15](#_Toc196227021)

[3.1.1. 아키텍처 한계점 15](#_Toc196227022)

[3.2. 최신 핵심 아키텍처 기술요소 및 구현 사례 16](#_Toc196227023)

[3.3. 핵심 아키텍처 사상 (텍스트 기반) 17](#_Toc196227024)

[3.3.1. 아키텍처 구조 17](#_Toc196227025)

[3.3.2. 아키텍처 사상 18](#_Toc196227026)

[3.4. 요구사항 변경관리 툴의 한계점에 대한 종합적인 시사점 (정리) 18](#_Toc196227027)

[3.4.1. 기술적 한계 측면 18](#_Toc196227028)

[3.4.2. 요구사항 변경관리 툴 도입 시 기술 과제 19](#_Toc196227029)

[3.4.3. 기업환경에서 요구사항 변경관리 툴 선택 시 고려 요소 19](#_Toc196227030)

[3.4.4. 요구사항 변경관리 툴 도입 시 미래 대응을 위한 전략적 시사점 20](#_Toc196227031)

[3.5. 기술 트렌드별 영향 요구사항 변경관리 툴 및 기술 연계 설명 20](#_Toc196227032)

[4. 요구사항 정의 23](#_Toc196227033)

[4.1. 요구사항 변경관리 툴 선택 요구사항 정의표 23](#_Toc196227034)

[4.2. 자가진단 체크 24](#_Toc196227035)

[4.3. 자가진단 기반 선택 가이드 25](#_Toc196227036)

[5. 도구 선정 가이드 27](#_Toc196227037)

[5.1. 선정조건 정리 27](#_Toc196227038)

[5.2. 요구사항 변경관리 툴 선정을 위한 평가기준 및 가중치 구성 28](#_Toc196227039)

[5.3. 요구사항 변경관리 툴 평가항목 점수표 29](#_Toc196227040)

[5.3.1. 평가항목 점수표 29](#_Toc196227041)

[5.3.2. 도구별 적용 전략 해석 29](#_Toc196227042)

[5.4. 프로젝트 유형별 · 상황별 · 규모별 맞춤 선정표 30](#_Toc196227043)

[5.4.1. 프로젝트 유형별/상황별 추천표 30](#_Toc196227044)

[5.4.2. 프로젝트 규모별 적용 가이드 31](#_Toc196227045)

[6. 부록 33](#_Toc196227046)

[6.1. 이해관계자 분석 33](#_Toc196227047)

[6.2. 요구사항 33](#_Toc196227048)

[6.2.1. 기능요구사항 33](#_Toc196227049)

[6.2.2. 비기능 요구사항 34](#_Toc196227050)

[6.3. 제약조건 35](#_Toc196227051)

[6.3.1. 기술적 제약조건 35](#_Toc196227052)

[6.3.2. 비즈니스 제약조건 35](#_Toc196227053)

[6.4. 솔루션 평가 36](#_Toc196227054)

[6.4.1. 평가영역 및 가중치 36](#_Toc196227055)

[6.4.2. 전략적 해석 가이드 36](#_Toc196227056)

[6.4.3. 상세평가 37](#_Toc196227057)

[6.4.4. 전체 평가 요약 39](#_Toc196227058)

# 아키텍처

본 문서는 전사 EA 표준 사업을 위한 솔루션 표준 프로파일을 도식화하고 상세에 대해 기술한 문서이다.

## 도입 목적

요구사항 변경관리툴은 단순히 요구사항을 적는 도구가 아닙니다. 전사 시스템과 아키텍처는 수많은 이해관계자와 복잡한 연계관계 속에서 운영되며, 여기에 요구사항이 수시로 변경되기 때문에 **통제되지 않은 변경은 품질 저하, 운영 리스크, 일정 지연으로 직결됩니다.**

따라서 이 도구는 다음과 같은 전사 전략적 목적을 위해 도입되어야 합니다:

* EA 구성 요소간 요구사항 추적 및 통제 체계 확보
* DevOps 및 ALM 연계 기반 하의 자동화된 변경 추적
* 프로젝트, 운영, 감사, 보안 등 조직 전 영역의 변경 요청 통합 관리
* 복잡한 시스템간의 요구사항 변경 이력, 영향도, 승인 흐름의 일관성 확보

|  |  |
| --- | --- |
| 구분 | 도입 목적 설명 |
| 전사 아키텍처 정합성 확보 | 요구사항 변경 시 EA 내 업무, 데이터, 애플리케이션, 인프라 계층 간 정합성을 유지하도록 변경 연동 및 영향도 파악 기반 확보 |
| 요구사항 변경 이력 통제 | 요구사항의 생성부터 변경, 승인, 반영까지 전체 이력과 상태 관리로 내부감사, 보안 대응, 추적성 확보 가능 |
| 프로젝트 간 재사용 및 중복 방지 | 유사 또는 동일 요구사항이 다른 프로젝트에서 반복되지 않도록 요구사항 등록/검색/참조 기능 구현 |
| 시스템 간 연계 요구사항 정합성 검증 | API, DB, 연계모듈 등 시스템 간 연계에서 발생하는 요구 변경의 일관성 유지 |
| DevOps 연계 자동화 기반 확보 | 요구사항 → 개발 → 테스트 → 배포까지 변경 내역 자동 연계 (Git, Jenkins, Test Tool 등과 통합) |
| 승인 기반의 변경관리 프로세스 정착 | 요구사항 변경 시 책임자 승인, 영향 분석자 검토, 개발자 및 테스터간 워크플로우 적용 |
| 표준화된 요구관리 메타데이터 체계 수립 | 요구사항의 유형, 출처, 우선순위, 영향도, 중요도 등을 메타데이터화하여 분석 및 분류 기반 마련 |
| ALM 및 QA 연동 기반 요구통제 | 요구사항과 테스트케이스, 결함, 검증 결과를 연동하여 품질 기반 릴리즈 대응 가능 |
| 모델 기반 아키텍처 연계 | 요구사항이 BPMN, UML, ERD 등의 모델 요소와 직접 맵핑되어 시각적 추적성 확보 |
| 지식자산화 기반의 전략적 요구통제 | 과거 요구사항의 변경 이력, 원인, 반영 결과 등을 분석하여 향후 요구관리 전략 수립 가능 |

요구사항 변경관리툴의 도입 목적은 단순히 도구 확보가 아닌, **요구사항이라는 핵심 자산을 EA 구조 하에 통제하고, 변경으로부터 시스템 품질을 보호하는 체계를 정착시키는 것**입니다.

특히 DevOps·QA·EA를 아우르는 연계 기반 통제 시스템으로 요구사항의 **수명주기 전체를 관리**하는 것이 핵심입니다.

|  |  |
| --- | --- |
| 시사점 | |
| 전사 IT 통제 | 요구사항은 곧 비즈니스 전략이며, 변경관리 없이는 시스템 안정성도 없다 |
| 아키텍처 정합성 | 요구사항 기반의 EA 계층 정합성 확보가 구조적 품질의 시작이다 |
| DevOps 연계 | 요구사항 기반으로 자동화된 개발-테스트-릴리즈 체계를 연결해야 한다 |
| 데이터 기반 운영 | 요구사항 변경 데이터를 축적해 미래 전략 수립과 운영 비용 절감에 활용 가능 |
| 확장성 | 조직 내 다양한 유형의 프로젝트 및 시스템 운영 시 공통 변경 통제 플랫폼으로 확장 가능 |

## 도입 전 선결과제

요구사항 변경관리툴은 단순한 기능성 도구가 아니라, **조직 전체의 변경 흐름을 통제하는** 핵심 **체계**입니다. 그러나 이러한 시스템을 성공적으로 도입하고 정착시키기 위해서는 다음과 같은 **기술적, 조직적, 프로세스적 선결 과제**를 사전에 해결해야 합니다.

많은 기업들이 툴 도입에 실패하는 이유는 기술적인 적합성보다는 **기존 조직문화, 관리 프로세스 부재, 이기종 시스템 연계 미비**와 같은 **비기술적 저항 요인** 때문입니다. 따라서 도구 선정 이전에 반드시 선결해야 할 과제를 다음과 같이 체계적으로 정리합니다.

|  |  |
| --- | --- |
| 항목 | 세부내용 |
| 요구사항 관리 프로세스 부재 | 변경 요청 수집 → 승인 → 반영 → 추적 → 릴리즈까지 전사 공통의 요구사항 변경 프로세스가 수립되어 있지 않음 |
| 역할 및 책임(R&R) 미정의 | 요구사항 생성자, 변경 승인자, 영향도 분석자, 품질검증자 등의 역할과 책임이 명확하지 않음 |
| 요구사항 메타데이터 정의 미흡 | 요구사항 우선순위, 출처, 영향범위, 변경 사유 등 분류 체계가 조직적으로 합의되지 않음 |
| 현행 시스템 연계 미흡 | Jira, Git, Jenkins, Test Tool 등과의 연계성 사전 검토 없이 도구만 도입될 경우 단절된 체계로 전락 가능 |
| 시스템/부서간 요구사항 중복 방지 미비 | 동일하거나 유사한 요구사항이 프로젝트 단위로 중복 발생 → 이력 및 참조 체계 미비 |
| 기존 툴들과의 기능 충돌 우려 | 기존의 산발적 요구사항 관리 수단(Excel, Word, Confluence 등)과 충돌 가능성 사전 제거 필요 |
| EA 체계와의 맵핑 미정의 | 요구사항과 업무프로세스(BPM), 데이터(ERD), 애플리케이션 구성도 간 연결 로직이 부재함 |
| 사용자/실무자 수용성 확보 필요 | 도구 도입이 업무 부담을 가중시킬 것이라는 인식 → 실 사용자 중심의 사전 교육 및 체계 마련 필요 |
| 변경 승인 및 영향도 분석 체계 미비 | 변경 승인 플로우, 영향도 시각화 기준, 변경사항 알림/이력 저장 기준 정립 필요 |
| 요구사항 라이프사이클 통합 정의 필요 | 수집-분석-승인-반영-검증-종료까지의 상태 흐름을 표준화하고 상태기반 통제 정책 마련 필요 |

요구사항 변경관리툴을 성공적으로 도입하기 위해서는 **기술 도입 이전에 조직 내 "변경관리 체계", "요구사항 프로세스", "역할 책임", "연계 기반"에 대한 통합 정리**가 선행되어야 합니다. 이러한 선결 과제가 해결되지 않으면 도입된 툴은 단순 문서 저장소 이상의 역할을 하지 못하고, **운영 부담과 저항만 커지게 됩니다.**

|  |  |
| --- | --- |
| 시사점 | |
| 요구사항 변경은 조직 전반의 통제 이슈다 | 단순히 개발자만의 문제가 아닌, 업무·운영·보안·감사까지 연결되는 관리체계이다 |
| 툴보다 중요한 것은 프로세스다 | 아무리 우수한 툴이라도, 전사 요구사항 관리 체계가 없으면 도입 효과는 낮다 |
| 사용자 수용성과 연계 가능성이 핵심이다 | 기존 도구, 연계 시스템과의 통합성, 실 사용자 중심의 운영 구조를 설계해야 한다 |
| 선결과제 해결 없이는 실패 확률 높음 | 기존 관행(문서, 메일, 엑셀)에 묶여 있는 실무 환경을 체계화하지 않으면 도입은 실패한다 |

## 핵심 아키텍처 기술 요소

요구사항 변경관리툴은 단순한 데이터 저장 시스템이 아닌, **요구사항 수명주기 전반을 통합·자동화·시각화·추적 가능한 구조로 지원하는 통합 아키텍처 체계**를 기반으로 설계되어야 합니다. 또한, 최근의 도구들은 단일 솔루션이 아닌 다양한 엔터프라이즈 시스템과의 \*\*연계 구조(API, ALM, DevOps 등)\*\*를 중심으로 구성되어 있으며, **EA, DevOps, 테스트 관리, 버전 관리 시스템 등과 연동 가능한 유연한 아키텍처**가 핵심 경쟁력입니다.

|  |  |
| --- | --- |
| 구성요소 | 기술요소 설명 |
| 요구사항 저장소 (Requirement Repository) | 모든 요구사항은 메타데이터 기반 구조화된 저장소에 저장되며, 생성자, 우선순위, 상태, 변경이력, 버전 등을 관리함 |
| 이력 및 버전관리 엔진 | 변경 발생 시 생성되는 각 버전에 대한 이력 추적, 롤백, 변경 승인 히스토리 제공. Git 및 ALM 툴과의 연계도 포함됨 |
| 요구사항 추적 매트릭스(Traceability Matrix) | 요구사항 ↔ 설계요소 ↔ 코드 ↔ 테스트케이스 ↔ 결함까지 상호 연결되는 추적 가능성 확보. 시각화와 영향도 분석 기능 포함 |
| 워크플로우 및 승인 프로세스 엔진 | 요구사항의 생성, 분석, 승인, 변경, 폐기까지의 상태 흐름과 역할(R&R)을 기반으로 승인 라인 구성 가능 |
| ALM 및 DevOps 통합 인터페이스 | Jenkins, Azure DevOps, GitHub, Jira 등과 연계하여 요구사항 기반 개발-테스트-릴리즈 연속 흐름 구현 가능 |
| 모델링 연계 기능 | BPMN, UML, ERD 등 EA 모델링 도구와의 연계 기반으로 요구사항의 위치와 연관성을 시각화함 (예: DOORS + Rhapsody) |
| 협업 및 리뷰 관리 기능 | 요구사항에 대한 리뷰, 코멘트, 토론, 상태기반 알림 등 실시간 협업 기능 내장 (예: Jama의 실시간 리뷰 모듈) |
| 변경 영향도 분석 모듈 | 요구사항 변경 시 하위 연관 요소에 대한 영향도 분석 자동화. 추적 매트릭스 기반으로 시스템 영향 예측 가능 |
| 통합 레포팅 및 감사 기능 | 요구사항 수명주기 리포트, 감사 로그, 승인 이력, 변경 사유 등 다양한 기준으로 정형화된 보고서 생성 |
| 보안 및 권한 관리 체계 | 사용자/역할 기반의 읽기/쓰기/승인 권한 분리, 이력 감사 및 외부 감사 대응을 위한 보안 설계 포함 |
| 확장 가능한 API 기반 구조 | 외부 시스템과의 연동을 위한 RESTful API, Webhook, OpenAPI 연계 구조를 기본적으로 탑재 |

요구사항 변경관리툴의 아키텍처는 단순한 요구 저장소가 아닌, 요구사항 기반으로 전사 시스템을 통제하고 EA, DevOps, QA, 형상관리까지 연결되는 중심 통합체계입니다.

특히, 요구사항-테스트-릴리즈-운영간 자동 연계를 위한 구조적 준비가 아키텍처 핵심이며, 이러한 기술요소 없이 도입된 도구는 단순 문서 작성 툴로 전락할 위험이 큽니다.

|  |  |
| --- | --- |
| 전략적 시사점 | |
| 아키텍처 중심 사고 필요 | 요구사항 관리 도구는 단순 기능이 아니라 전사 시스템의 통제 흐름을 설계하는 플랫폼이다 |
| 연계 기반 설계 필수 | ALM, DevOps, QA, 형상관리 도구와의 통합 인터페이스가 구축되어야 전사 활용 가능 |
| 추적성과 영향도 분석 | 변경된 요구사항이 어디에 영향을 주는지 실시간으로 분석하고 반영하는 체계가 반드시 필요 |
| 확장성과 보안성 고려 | 권한관리, 감사로그, API 확장성 등 전사 정보보안 및 통제 기준에 맞춘 설계가 요구됨 |
| 도입 전 연계 시나리오 정립 필요 | 툴 중심이 아닌 아키텍처 중심의 연계 시나리오를 기반으로 도입/설정 전략이 수립되어야 함 |

## 핵심 특징 및 장단점 비교

요구사항 변경관리툴을 도입할 때 가장 중요한 것은 단순히 기능이 풍부한 도구를 선택하는 것이 아니라, **조직의 아키텍처 전략, 프로세스 성숙도, 연계 시스템, 개발 방식(DevOps/ALM)과 얼마나 정합성을 갖고 통합될 수 있는지**를 평가하는 것입니다.

본 절에서는 EA 연계, DevOps 호환성, 요구사항 추적성, 사용자 협업 기능, 확장성, 유지보수 편의성 등을 기준으로 국내외에서 대표적으로 사용되는 **7개 도구**에 대해 **핵심 특징, 장점, 단점**을 비교 분석합니다.

|  |  |  |  |
| --- | --- | --- | --- |
| 툴명 | 주요특징 | 장점 | 단점 |
| IBM DOORS / DOORS Next | 대규모 요구사항 및 변경관리 전용 플랫폼, ReqIF 기반, Rhapsody/BPMN 연동 | 전통적인 엔터프라이즈 요구사항 관리에 강력, 자동화된 추적 매트릭스, 영향도 분석, 수명주기 관리 완전 지원 | UI 복잡, 학습 곡선 큼, 사용자 친화성 부족, 구축 비용 높음 |
| Jira (with Structure, Xray, Zephyr) | 애자일 기반 이슈·요구사항·테스트 통합 관리, Atlassian 생태계와 연동 가능 | 개발자 친화적, 유연한 커스터마이징, DevOps·Git·CI/CD 연계 용이, 실사용자 도입 확산 빠름 | 요구사항 관리 기능은 플러그인에 의존, 복잡한 변경 승인·추적에 한계 |
| Jama Connect | 요구사항·테스트·위험관리 통합 플랫폼, 실시간 협업 중심 구조 | 직관적인 UI/UX, 실시간 협업 리뷰, 규제 기반 산업(자동차, 의료) 특화, API 연계 우수 | 고비용, IT 인프라와 연동 시 전문성 요구, 중소규모에 과도할 수 있음 |
| Helix RM (Perforce) | ALM 기반 변경관리, 버전관리/요구사항/테스트 통합 | 요구-테스트-결함 통합이 우수, ALM 툴 연동 최적화, 완전한 변경 이력 관리 | 설정 복잡도 높음, 국내 사용자 적음, 온프레미스 중심 |
| Modern Requirements4DevOps | Azure DevOps 기반 요구사항 확장 모듈, Office Word 기반 문서화 | MS DevOps와의 자연스러운 통합, 문서 중심 요구관리, 워크아이템 연동 우수 | Azure DevOps 미사용 시 도입 어려움, 국내 도입사례 희소 |
| CodeBeamer (Intland) | ISO26262, ASPICE, FDA 대응 가능한 ALM+RM 통합도구 | 품질보증·규제관리 중심 조직에 특화, 요구사항-테스트-릴리즈 완전통합 | 초기 비용 부담, DevOps와 연계는 약간 부족, 사용성은 중간 수준 |
| Polarion (Siemens) | Siemens의 ALM+요구관리, 모델링 연계 특화 | 요구-테스트-결함-릴리즈 전체 프로세스 커버, BPMN/UML 모델과 연결 우수 | 인터페이스 무거움, 중소기업에는 과도함, CI/CD 통합은 한계 있음 |

요구사항 변경관리툴은 기술 사양만이 아닌, **조직의 시스템 생태계, 도입 목적, 아키텍처 정합성, 운영 주체의 성숙도, DevOps 연동 가능성** 등 복합적 관점에서 선정되어야 합니다.

툴 간 비교를 통해 도출되는 인사이트는 기능 중심이 아닌, **전략·운영·확장·연계 중심의 의사결정 기준을 마련**하는 데 목적이 있습니다.

|  |  |
| --- | --- |
| 시사점 | |
| 아키텍처 정합성 기반의 툴 도입 필수 | 도입하려는 툴이 기존 EA 구조와 통합될 수 있는 아키텍처 구조를 갖춰야 함 |
| 단일기능보단 연동력 중심으로 판단 | Jira와 같은 개발기반 도구는 연동에 유리하나, 요구추적 기능은 별도 보완 필요 |
| 규제·품질 기반 산업은 전문 툴 고려 | DOORS, CodeBeamer, Polarion은 규정 대응에 유리하나, 도입 비용과 운영 인력 고려 |
| 협업 및 UI는 사용자 도입속도 좌우 | Jama, Jira는 UI와 협업이 강점으로 빠른 확산 가능성 높음 |
| 플러그인 중심 도구는 범위 명확화 필요 | Jira 기반 구조는 플러그인 의존도가 높아, 조직의 변경관리 범위 정의 필수 |

# 시장 상황

본 문서는 전사 EA 표준 사업을 위한 솔루션 표준 프로파일을 도식화하고 상세에 대해 기술한 문서이다.

## 시장 동향 및 기술 연계 영향도

### 시장 상황 및 동향

요구사항 변경관리툴은 **ALM(Application Lifecycle Management), DevOps, 모델링 연계, 테스트 및 품질관리** 기능과 연계되는 고도화된 관리 시스템으로 진화하고 있습니다. 과거에는 단순 문서형 툴(Word, Excel, Wiki 등)을 활용하던 기업들도, **지속적인 감사, 보안, 품질 규제 대응, 글로벌 협업 증가**에 따라 툴 기반의 자동화된 변경관리 체계 도입을 본격화하고 있습니다.

시장 흐름은 다음과 같이 요약됩니다:

* **대형 엔터프라이즈 시장**은 DOORS, CodeBeamer, Polarion 중심
* **애자일 및 DevOps 시장**은 Jira, Modern Requirements 중심
* \*\*고품질 규제 산업(자동차, 헬스케어, 항공)\*\*은 Jama, CodeBeamer, Helix RM이 우세
* **UI/UX 및 협업 중심 중견기업**은 Jama, Jira 중심 확산

|  |  |  |  |
| --- | --- | --- | --- |
| 구분 | 국내 시장 상황 | 글로벌 시장 상황 | 주요 시장 동향 |
| 도입 단계 | * 금융, 제조, 공공 중심으로 점진적 확산 * 대기업 계열사 주도 하에 전사 적용 중 | * 북미·유럽은 ALM/DevOps 통합기반의 완성도 높은 시장 형성 | * RM(Requirements Management) + ALM 통합화 확산, DevOps 연계 강화 |
| 주요 채택 툴 | * Jira + 플러그인 (Structure/Xray) * IBM DOORS, Polarion 일부 | * IBM DOORS, Jama, CodeBeamer, Polarion, Helix RM * Jira 기반 확대 | * Jira의 생태계 확장 + 전통 요구도구의 DevOps 통합화 |
| 경쟁 구도 | * 개발자 기반은 Jira 중심, 품질·감사 요구 많은 조직은 DOORS 계열 | * 협업성과 사용자 UI는 Jama, 규제/품질 대응은 CodeBeamer/POLARION 선호 | * SaaS 기반 API 연계/확장성 높은 도구 우세 |
| 전개 방식 | * 사내 IDC 설치형 + 점진적 SaaS 전환 | * 대부분 클라우드 SaaS 기반 + API 연계 플랫폼 구조 | * 클라우드 우선 전략, 오픈 API 중심 연계 구조 확산 |
| 요구 유형 | * 기능 요청 + 업무 연계 요구 + 문서 추적 | * 요구사항-테스트-릴리즈 일체형 요구 증가 | * 수명주기 통합 관리 기반으로 확산 (ALM+RM+QA) |
| 기술 연결성 | * EA/PMO 연계는 약함, DevOps는 부분 연동 | * Git, Jenkins, Azure DevOps, TestRail 등과 통합 구조 활발 | * API 기반 도구 간 연결 우수, 확장성 중심 기술 요구 증가 |

요구사항 변경관리 시장은 단순 관리 도구에서 벗어나, **조직 전체의 소프트웨어 생명주기, 품질 통제, 감사 이력, 협업 구조를 자동화하는 통합 플랫폼**으로 발전하고 있습니다.

특히, 글로벌 시장은 **모델 기반 요구사항 관리, API 중심 연계성, 규제 대응, 실시간 협업성**을 기반으로 한 차세대 RM 도구들이 주도하고 있으며, 국내 시장도 빠르게 이 흐름을 따라가고 있습니다.

|  |
| --- |
| 시사점 |
| 전사 RM 체계는 단일 기능이 아닌 통합 플랫폼 중심으로 확산되고 있음 |
| 국내 기업도 점진적으로 DevOps, ALM, 모델링 도구와 연계되는 방향으로 이동 중 |
| 글로벌 시장은 SaaS, API, 실시간 협업 중심 도구로 재편 중이며, 보안·감사 기능 강화 흐름도 함께 확산 |
| 툴 선택 시 UI/UX뿐 아니라 기술연계성, 조직 구조 대응력을 함께 평가해야 함 |

### 기술동향 영향도 평가

요구사항 변경관리툴은 단순한 "문서 저장소"에서 벗어나, **"아키텍처 중심 자동 추적 체계", "CI/CD 통합 흐름", "실시간 협업", "모델 기반 요구관리(MBSE)"** 등 다양한 최신 기술 패턴과 직접 연동되는 구조로 진화하고 있습니다.

특히, 기술동향별로 각 툴이 얼마나 수용/연계 가능한지는 **툴의 생존력과 전략적 가치**에 직결됩니다. 다음은 기술 트렌드와 툴 간 영향도 매핑을 정리한 표입니다.

|  |  |  |  |
| --- | --- | --- | --- |
| 기술동향 | 툴명 | 영향도 (강/중/약) | 이유 설명 |
| DevOps 자동화 연동 (CI/CD) | Jira, Modern Requirements | **강** | Jira는 Git/Jenkins/Azure DevOps와 원활한 통합, Modern은 Azure DevOps 연동 최적화 |
| ALM 통합 연계 | Helix RM, CodeBeamer, Polarion | **강** | 요구-테스트-릴리즈 통합 제공, ALM 기반 품질보증 체계 우수 |
| 모델 기반 요구사항 관리 (MBSE) | DOORS, Polarion | **강** | BPMN, UML, SysML 등 모델과 요구사항 직접 연계 가능 |
| 실시간 협업 / 사용자 UX | Jama, Jira | **강** | Jama는 리뷰/댓글/알림 실시간 처리, Jira는 사용자 편의 중심 UI 제공 |
| 표준 기반 교차 플랫폼 (ReqIF) | DOORS, ReqIF.ac | **중** | 국제 표준 기반 요구사항 포맷으로 타 도구와 상호운용 가능 |
| 보안 및 감사 대응 기능 | CodeBeamer, Polarion | **중** | 감사로그, 규제 대응, 변경 이력 고정화 구조 탑재 |
| AI 기반 요구 분석 지원 | 없음 (Emerging) | **약** | 현재는 미성숙 영역, 일부 R&D 수준 도입 중 (예: 요구사항 자동 분류 AI) |

기술 트렌드와의 연계성은 도구의 장기 활용성과 확장성을 결정하는 핵심 요소입니다.  
**단순 기능 중심 평가를 넘어서, 변화하는 기술환경 속에서 도구가 어떻게 진화·확장 가능한지를 판단해야** 진정한 도입 ROI를 확보할 수 있습니다.

|  |
| --- |
| * 툴 도입 시 단기 기능이 아닌 장기 기술 연계성을 반드시 평가해야 함 * DevOps, ALM, MBSE 등 기술 트렌드 수용력이 높은 도구가 생존력과 투자 효율이 높음 * 글로벌 확산 속도에 비해 국내 도구는 기술 연계성 준비 부족 → 사전 시나리오 수립이 중요 * 기업은 내부 개발환경과 기술로드맵에 따라 툴별 영향도/호환성 시뮬레이션을 먼저 수행해야 함 |
|
|
|

## 시장 점유율 및 주요 기업 채택 사유

요구사항 변경관리툴의 시장 점유율은 단순 도입률이 아닌, **조직의 산업군, 품질 요구 수준, 통제 체계 성숙도, 기술 아키텍처 연계성** 등 다양한 요소에 의해 결정됩니다.

본 항목은 **다양한 산업군에서의 채택 사례를 기반으로 실질적인 도입 동기와 채택 목적**을 정리하며, EA 아키텍처 연계 기준에서의 툴 적합성 및 도입 가치 판단 기준을 제공합니다.

|  |  |  |  |
| --- | --- | --- | --- |
| 툴명 | 시장점유율 (추정) | 주요 채택 기업 (산업군) | 채택사유 요약 |
| IBM DOORS / DOORS Next | * 글로벌 약 35% * 국내 약 12~15% | 방위산업체, 금융 대기업, 공공기관 | * 요구사항 이력 추적/감사 대응 필수 환경에서의 고신뢰 체계 확보 목적 * RFI/RFP 단계부터 RM 구조 명세 요구되는 고규제 시장에 필수 |
| Jira (with Structure, Xray) | * 글로벌 약 45% * 국내 약 60% | IT 서비스, 핀테크, 중견 제조사 | * DevOps 기반 개발문화에 최적화된 환경 구성 * 개발-테스트-운영 간 유기적 연계와 플러그인 활용 유연성 |
| Jama Connect | * 글로벌 약 8~10% * 국내 제한적 | 글로벌 전기차 제조사, 헬스케어 장비사 | * 협업 기반 요구 리뷰, 의료/자동차 안전 기준(FDA, ISO26262 등) 대응 목적 |
| CodeBeamer | * 글로벌 약 6~8% * 국내 5% 내외 | 전장부품사, 제약 품질관리부서 | * 규제 산업 대응 및 테스트-릴리즈 추적 강점, ASPICE/FDA 인증 필요 환경에서의 채택 증가 |
| Modern Requirements4DevOps | * 글로벌 약 3% * 국내 미미 | 일부 북미 MS DevOps 고객사 | * Azure DevOps 전용 워크아이템 기반 조직 중심 확산 MS 생태계 중심 조직에서 빠른 내재화 가능 |
| Polarion | * 글로벌 약 4~6% * 국내 대형 제조사 일부 | 항공전자, 철도 시스템 기업 | * 시스템 엔지니어링 기반의 모델 연계/릴리즈 관리 통합 * 기반 구조의 복잡성을 수용할 조직 성숙도 전제 필요 |
| Helix RM (Perforce) | * 글로벌 약 3% * 국내 미진입 | 미국 방산, 엔지니어링 그룹사 | * 완성도 높은 ALM 통합 기능과 고성능 릴리즈 추적 체계 구축 목적 |

요구사항 변경관리툴의 시장점유율은 단순히 툴의 완성도보다는 **도입 주체의 산업 특성, 내부 통제 요구 수준, 기술 인프라, 조직의 성숙도**에 따라 명확히 분화되고 있습니다.

따라서, 툴 자체만을 비교하는 것이 아니라 **“우리 조직이 이 툴이 요구하는 통제 수준을 감당할 수 있는가”에 대한 내재화 관점의 판단**이 선행되어야 합니다.

|  |
| --- |
| * 산업 특화 도구는 규제 대응·감사 통제 기반에서 절대적 역할 수행 * 기술 유연성 중심의 도구는 빠른 전사 확산이 가능하나, 통제 범위는 명확히 설정해야 함 * 조직 성숙도에 따라 툴의 정착 가능성은 상이하므로, 도입보다 정착 전략이 핵심 * 툴 도입은 전사 기술·운영 전략과 병행되어야 하며, 단독 운영은 리스크를 초래할 수 있음 |
|
|
|

## 주요 요구사항 변경관리 툴 완성도 평가

툴의 채택 여부를 결정짓는 가장 핵심적인 요소는 **기능적 완성도, 기술 연계성, 운영 유연성, 사용자 수용성**입니다. 단순히 "기능이 있다"는 수준이 아닌, **각 기능이 조직의 아키텍처, 운영 프로세스, 인력 체계와 어떻게 유기적으로 작동할 수 있는지**에 대한 완성도 평가가 중요합니다.

본 항목에서는 다음 기준으로 주요 툴들을 **수치화된 평가 기준**으로 분석합니다:

* **기능적 적합성 (Functional Fit)**
* **기술적 통합성 (Architectural Integration)**
* **운영 효율성 (Operational Readiness)**
* **사용자 경험 및 확장성 (UX/Scalability)**
* **전사적 활용성과 전략 정합성 (Enterprise Adaptability)**

### 도구별 완성도 평가표

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 평가항목 | 설명 | | IBM DOORS | Jira (w/ 플러그인) | Jama Connect | CodeBeamer | Polarion |
| 기능적 적합성 | 요구관리, 변경관리, 추적성의 기능적 완성도 | | 9.5 | 7.0 | 8.5 | 9.0 | 8.8 |
| 기술적 통합성 | DevOps, ALM, 모델링 도구와의 연동성 | | 8.5 | 9.2 | 8.0 | 8.8 | 9.0 |
| 운영 효율성 | 초기 설정, 워크플로우 구성, 유지관리 용이성 | | 7.0 | 9.5 | 8.3 | 7.5 | 7.8 |
| 사용자 경험/확장성 | UI/UX, 협업 기능, 확장 API 제공 | | 6.8 | 9.4 | 9.5 | 7.8 | 7.2 |
| 전사적 정합성 | EA 구조 내 연계성, 정책 대응 유연성 | | 9.2 | 7.5 | 8.6 | 9.0 | 9.1 |
| 점수 범위: 1점(미흡) ~ 10점(우수) | | | | | | | |
| 점수 | | **기준 설명** | | | | | |
| 10점 | | 업계 표준 수준의 완성도, 자동화 및 확장성 최상 | | | | | |
| 9점 | | 주요 기능 탑재 및 일부 고급 기능 제공 가능 | | | | | |
| 8점 | | 범용적인 활용 가능, 다소 부족한 요소 있음 | | | | | |
| 7점 | | 기능은 있으나 복잡성, 학습 필요성 존재 | | | | | |
| 6점 | | 일부 통합 또는 사용자 기능 제한됨 | | | | | |
| 5점 이하 | | 사용 목적 제한적, 대체제 고려 필요 | | | | | |

### 평가 항목별 점수 기준표

|  |  |
| --- | --- |
| 평가항목 | 기준 설명 |
| 기능적 적합성 | 요구·결함·테스트 연계 / 변경 이력 추적 / 승인 워크플로우 내장 / 영향도 분석 등 내재화 수준 |
| 기술적 통합성 | Git, Jenkins, Azure DevOps, 모델링 툴, ALM 도구와의 통합성 (API, Webhook, ReqIF 등 포함) |
| 운영 효율성 | 사용자 권한 구성, 워크플로우 설정, 데이터 관리, 템플릿 제공, 설치·운영 복잡도 등 |
| 사용자 경험 및 확장성 | 실사용자 중심 UI/UX, 실시간 협업, 노코드 커스터마이징, 레포팅 기능, 확장 인터페이스 |
| 전사적 정합성 | EA, 프로젝트관리, 품질/감사 기준, 통제 프레임워크와의 연계성 수준 및 확장성 |

툴의 도입 성공 여부는 "가장 많은 기능을 가진 도구"가 아니라 \*\*"우리 조직이 가장 잘 흡수하고 통합 운영할 수 있는 도구"\*\*를 선정하느냐에 따라 달라집니다.

완성도 평가는 단순 점수가 아니라, **조직의 현재 수준과 미래 로드맵을 반영한 전략적 적합성 평가의 도구**로 활용되어야 합니다.

|  |
| --- |
| * 완성도는 단순 기능 집합이 아니라, 아키텍처 내 통합 운영 관점에서 평가되어야 한다 * 높은 기술 연계성 + 낮은 UI 친화성은 정착율을 떨어뜨릴 수 있다 (예: DOORS) * 개발자 기반 조직은 Jira 기반이 도입에 유리하지만, 감사 통제에는 한계가 있다 * Jama, CodeBeamer, Polarion은 품질관리·규제대응 목적일 경우 전략적 선택지가 될 수 있다 * 평가점수는 조직 성숙도와 연계 전략에 따라 재해석되어야 한다. 일률적 점수는 위험하다 |

## 기술 트렌드에 따른 영향도 및 이유

요구사항 변경관리툴은 최근 **디지털 전환 가속화, DevOps/ALM 통합 흐름, 모델 기반 시스템 엔지니어링(MBSE)** 등의 기술 흐름과 밀접하게 연결되고 있습니다. 단순한 기능 여부가 아닌, **현재와 미래 기술 트렌드에 대해 해당 도구가 얼마나 유기적으로 적응하고 있는가**가 전략적 도입 판단의 핵심이 됩니다.

본 항목에서는 주요 기술 트렌드별로 각 툴이 **얼마나 강하게 수용하고 있는지, 그 영향도는 무엇인지, 조직 전략에 어떤 시사점을 주는지**를 분석합니다.

|  |  |  |  |
| --- | --- | --- | --- |
| 기술동향 | 툴명 | 영향도 | 이유 설명 |
| DevOps 통합 연계 (CI/CD, GitOps) | Jira, Modern Requirements | **강** | Jira는 Git, Jenkins, Bitbucket, Azure DevOps 등과의 통합 연계성이 탁월하고, REST API 활용이 자유로움. Modern은 Azure DevOps 생태계에 최적화 |
| ALM 수명주기 통합 | CodeBeamer, Polarion, Helix RM | **강** | 요구사항 → 테스트 → 결함 → 릴리즈로 이어지는 흐름을 내장 워크플로우와 UI로 통합 관리 가능. 품질보증과 ISO26262 같은 규제 대응에 최적화 |
| MBSE 및 모델 연계 아키텍처 | DOORS, Polarion | **강** | SysML, BPMN, UML 기반 모델링 툴(Rhapsody 등)과의 연계가 가능하고, 모델 객체 기반 요구 매핑 기능을 보유 |
| 협업 기반 사용자 경험 (UX) | Jama, Jira | **강** | 실시간 리뷰, 댓글, 변경 알림 등의 기능을 통해 사용자와 이해관계자 간의 협업 및 요구 검토가 용이함. Jira는 애자일 협업 중심, Jama는 품질 중심 협업 |
| SaaS 및 클라우드 확장성 | Jama, Modern Requirements | **중** | 클라우드 기반으로 구축되어 운영 부담이 적고, 빠른 배포와 글로벌 협업이 용이함. 단, 온프레미스 연계 시 일부 제약 존재 |
| 보안 및 변경이력 감사 | DOORS, CodeBeamer | **중** | 감사 대응, 변경로그 이력 통제, 승인 기반 워크플로우 지원이 강력하여 외부감사 및 규제 대응 환경에 적합 |
| AI 기반 요구 자동화/분류 | N/A *(Emerging)* | **약** | 일부 툴(Jama, Polarion)은 AI 기반 요구 유사성 탐지, 자동 분류 기능을 실험 중이나, 상용화 단계는 아님 |
| ReqIF/OSLC 기반 도구 간 표준 연계 | DOORS, Polarion, CodeBeamer | **중** | 타 벤더 간 도구 연계(예: ALM-QA 도구, 모델링 도구) 시 표준 포맷 기반 연동이 가능하여 대규모 다벤더 환경에 적합 |

기술 동향에 대응하는 도구의 능력은 그 자체로 **투자 대비 유지·확장 가능한 생명력**을 의미합니다. 단순한 도입 평가 항목이 아니라, **향후 기술 스택 변화에 따라 도구가 어떻게 진화 가능한지에 대한 기술 지속 가능성 평가**로 활용해야 합니다.

특히, **DevOps, ALM, MBSE 통합**은 단일 도구가 아닌 전사 시스템 연계 전략의 중심 축이 되므로  
**툴의 기술 영향도는 EA 아키텍처에서 직접 검토되어야 할 요소**입니다.

|  |
| --- |
| * 기술 트렌드와의 연계력은 도구의 생존 기간을 결정짓는 핵심 지표이다 * DevOps 및 ALM 통합을 고려하는 조직이라면 Jira, CodeBeamer, Polarion 중심으로 전략 수립이 필요하다 * MBSE 기반 EA 체계를 준비 중이라면 DOORS 또는 Polarion과 같은 모델 연계 기반 도구를 선제적으로 고려해야 한다 * 단기 ROI보다는 장기 아키텍처 변화에 대한 유연성을 가진 도구가 전략적 선택지다 * 향후 AI, 자동화 기반 기능의 성숙도에 따라 Jama, Polarion 등의 기술 경쟁력은 한층 강화될 것으로 보인다 |

# 기술 동향

본 문서는 전사 EA 표준 사업을 위한 솔루션 표준 프로파일을 도식화하고 상세에 대해 기술한 문서이다.

## 전통적인 요구사항 변경관리 툴(도구)들의 한계점과 시사점

과거의 요구사항 변경관리툴은 주로 **문서 기반 저장, 승인 기록 보관, 수작업 추적 관리**에 초점이 맞춰져 있었습니다. 이는 일정 수준의 변경 이력 통제는 가능했지만, **EA 아키텍처 연계, DevOps 통합, 실시간 협업, 규제 대응, 다변화된 연계 시스템**과 같은 **현대적 IT 생태계에서는 근본적인 기술 한계와 비효율성**을 드러내고 있습니다.

### 아키텍처 한계점

|  |  |
| --- | --- |
| 구분 | 한계점 |
| 정적 저장 기반 구조 | 요구사항은 단순 필드와 메모 수준으로 저장되며, 구조적 메타데이터 및 관계형 요구 추적 기능이 부재. 추적 매트릭스 구성은 외부 수작업으로 별도 관리됨 |
| 수동 중심 승인 체계 | 변경 승인 절차가 워크플로우 기반이 아니라 이메일, 문서 기반으로 수행되어 통제 근거가 약하며, 승인자 간 병렬성/역할 체계 불명확 |
| 연계 기반 아키텍처 미흡 | 외부 시스템(API, DevOps, 형상관리 등)과의 연결이 구조적으로 불가능하거나, 사용자 정의 스크립트에 의존해야 함 |
| 협업 기능 미비 | 요구사항 리뷰, 토론, 의견 기록, 변경 알림 등의 실시간 협업 기능이 없어 PM/개발자/테스터 간 커뮤니케이션이 단절됨 |
| 변경 이력의 단편적 보존 | 버전 이력은 있지만, 비교, 복원, 변경 사유 분석, 영향도 시각화가 되지 않아 실질적 변경 통제 체계로 활용 불가 |
| 테스트/결함과의 연계 없음 | 요구사항 ↔ 테스트 케이스 ↔ 결함 사이의 구조적 연결이 없어, 품질 추적/검증 흐름을 연결할 수 없음 |
| DevOps/CD 연동 불가 | Jenkins, Git, Azure DevOps 등과의 자동화 연계가 어렵고, 변경 요구가 릴리즈로 자동 흐르지 않음 |
| 감사/규제 대응 한계 | 승인 이력, 변경 로그, 통제 기준 등 외부감사 또는 품질 인증 대응을 위한 자료 자동화가 되지 않음 |
| 단일 프로젝트 중심 | 프로젝트 범위 내의 요구사항만 관리되며, 전사 공통 요구사항 자산화 및 재활용 체계가 없음 |
| 시각화 및 영향도 분석 부재 | 요구사항 간 관계도, 하위 영향도, 시스템 구조에 대한 시각적 연계성이 없어 EA 맵핑 및 시스템 영향 분석 불가 |

전통적인 요구사항 관리 체계는 **시스템과 개발 방식의 복잡성이 낮았던 시대**에는 유효했지만, **지금의 멀티 프로젝트, 규제 환경, DevOps 중심 개발 조직**에는 한계가 명확히 드러납니다.

특히, 변경 통제 체계를 수립하려는 조직에게는 **“형식적 요구사항 저장소” 수준에서 “전사 전략 통제 플랫폼”으로의 구조적 전환**이 반드시 필요합니다.

|  |
| --- |
| * 전통적 툴은 변경 통제보다는 기록 목적에 가까우며, 실질적 통제 체계 수립에는 부적합 * 전사 EA 체계와 연결되지 않은 요구사항 도구는 전략적 통합 통제 수단으로 기능할 수 없음 * DevOps, ALM, 테스트 연계 등 요구 기반 생명주기 흐름과 분리된 도구는 향후 조직 기술 성장에 제약 요소로 작용 * 단일 프로젝트 중심에서 벗어나 전사 범위의 요구 통합 관리체계를 구축할 필요가 있으며, 이는 툴 교체만으로는 해결되지 않음 * 툴 아키텍처는 단순 기능이 아닌, 아키텍처 통합성과 운영 내재화 전략의 중심축으로 재설계되어야 한다 |

## 최신 핵심 아키텍처 기술요소 및 구현 사례

최근의 요구사항 변경관리툴은 더 이상 단순 저장/이력 관리 수준에 머무르지 않습니다. **EA 구조, DevOps 파이프라인, 품질 시스템, 테스트·검증 체계, 외부 감사 및 규제 준수까지 연계된**   
\*\*“전사 통합 요구 흐름 관리 플랫폼”\*\*으로 진화하고 있으며, 핵심 아키텍처 요소는 다음 세 가지 중심축으로 정리됩니다:

1. **수명주기 연계 기반 구조 (Lifecycle-Integrated Architecture)**
2. **이벤트 기반 연동 아키텍처 (Event-Driven Sync)**
3. **지능형 협업 및 추적 아키텍처 (AI-assisted Traceability)**

이제 아래 표에서 각 주요 기술 요소별로 그 구조와 적용 사례, 구현 효과를 정리합니다.

|  |  |  |  |
| --- | --- | --- | --- |
| 구분 | 핵심 아키텍처 기술요소 | 적용 툴 사례 | 구현 효과 및 특징 |
| 추적성 기반 수명주기 통합 | 요구사항 ↔ 테스트 ↔ 코드 ↔ 결함 ↔ 릴리즈 간의 양방향 링크 자동 구성 (Traceability Matrix) | CodeBeamer, Polarion, Jama | 요구사항 변경 시 품질 및 테스트 영향 자동 분석, 전체 통합 이력 시각화, 요구-테스트-결함 연계 운영 가능 |
| DevOps 파이프라인 통합 구조 | Jenkins, Git, Azure DevOps, Jira, TestRail과의 REST API / Webhook 기반 통합 | Jira (Xray), Modern Requirements | 요구 생성 → 개발 Task 자동 생성 → 자동 테스트 → 결과 링크로 연결되는 통합 흐름, 릴리즈 가시성 극대화 |
| ReqIF/OSLC 기반 상호운용 연계 | 다양한 ALM, 테스트, 모델링 툴과의 표준 기반 데이터 포맷 연계 | IBM DOORS, Polarion, CodeBeamer | 툴 간 데이터 정합성 확보, 타 부서·협력사·외부 벤더 간 시스템 통합 운영 지원 |
| 실시간 협업 및 요구 승인 워크플로우 | 사용자 중심 승인, 변경 알림, 리뷰 및 이슈 커뮤니케이션 내장 | Jama Connect, Jira | 비즈니스·기술 사용자 간 실시간 변경 검토 및 승인, 협업 기반 품질 프로세스 정착 |
| 시각적 영향도 분석 | 요구사항 간 계층도/연계도/의존성 기반 영향 시각화 (Change Impact Visualization) | Jama, CodeBeamer, Polarion | 요구 변경 시 파급 대상 시스템/기능/테스트 자동 표시, 검토자 의사결정 정확도 향상 |
| 모델 기반 연계 (MBSE Architecture) | 요구사항 ↔ BPMN, UML, SysML 모델링 객체 매핑 및 추적 | DOORS + Rhapsody, Polarion | EA 기반 프로세스/시스템 구조 연계, 모델 중심 시스템 설계에서 요구 간 명확한 연결 확보 |
| 감사/규제 대응 구조화 | 변경이력, 승인로그, 릴리즈 이력 등 감사 대응 전용 레포팅 구조 내장 | CodeBeamer, DOORS | FDA, ISO26262, ASPICE 등 규제 산업에서의 사용성 검증, 외부감사 대응 문서 자동 생성 |
| AI 기반 요구 분석(도입 초기 단계) | 자연어 기반 유사 요구 매핑, 변경 제안 자동화, 자동 클러스터링 | Jama, Polarion(베타) | 요구 중복 방지, 분류 자동화, AI 기반 영향 범위 탐지 지원 *(R&D 영역)* |

최신 요구사항 변경관리툴은 단순히 요구를 ‘기록’하는 것이 아니라 **조직의 전략적 변화 흐름과 운영 절차 전체를 통제하는 중앙 허브 역할**로 진화하고 있습니다.

EA 관점에서는 이 툴이 단독 구성요소가 아니라, **DevOps, 테스트관리, ALM, 형상관리, 모델링 툴과 연계되는 구조적 중심축으로 작동해야 함**을 의미합니다.

|  |
| --- |
| * 최신 도구는 단일 기능이 아닌, 연계 흐름 중심의 구조적 플랫폼으로 진화하고 있음 * 단절된 기능이 아닌, 연계 중심의 데이터 흐름이 도구의 전략적 가치를 결정한다 * EA, DevOps, QA, 감사 시스템과의 연동 준비 없이 도입된 툴은 단순 저장소에 불과 * 조직은 툴 자체보다, 툴이 들어가는 아키텍처의 구조와 운용 시나리오를 먼저 설계해야 한다 * 향후 AI 기반의 요구 유사도 분석, 변경 패턴 예측 기술은 전략적 진화를 가속화할 것 |

## 핵심 아키텍처 사상 (텍스트 기반)

요구사항 변경관리 아키텍처는 단순히 기능을 연결하는 기술적 구조를 넘어, **조직의 전략 수립, 변화 대응, 품질 통제, 개발 생산성, 운영 안정성**을 동시에 실현하기 위한 **전사 통제 철학의 반영 구조**입니다.

따라서 요구사항 변경관리툴의 아키텍처는 단일 시스템 내부의 구성 개념이 아니라, **"요구의 흐름(Flow of Change)"을 중심으로 한 EA 구조상 전략적 연결 고리**로 바라보아야 합니다.

### 아키텍처 구조

|  |
| --- |
| 아키텍처 구조 |
| ┌────────────────────────────────────────────┐  │ [1] 요구는 단순한 “데이터”가 아니다 │  │ → 관계 중심의 구조체로 관리되어야 한다 │  └────────────────────────────────────────────┘  │  ▼  ┌────────────────────────────────────────────┐  │ [2] 요구의 흐름은 단절 없는 구조를 따라야 함 │  │ - 업무, 개발, 테스트, 운영까지 흐름이 일관돼야 함 │  └────────────────────────────────────────────┘  │  ▼  ┌────────────────────────────────────────────┐  │ [3] 요구 변경은 “단일 이벤트”가 아니다 │  │ → 영향 전파 구조(Impact Propagation)가 설계되어야 함 │  └────────────────────────────────────────────┘  │  ▼  ┌────────────────────────────────────────────┐  │ [4] 모든 요구는 조직의 전략과 연결돼야 함 │  │ - EA의 BPM ↔ APP ↔ DATA ↔ INFRA와 정합성 확보 │  └────────────────────────────────────────────┘  │  ▼  ┌────────────────────────────────────────────┐  │ [5] 통제는 “자동화된 협업” 구조 위에서만 가능 │  │ - 실시간 승인, 알림, 이력, 테스트, 릴리즈 연동 구조 │  └────────────────────────────────────────────┘  │  ▼  ┌────────────────────────────────────────────┐  │ 🔁 [결론] 요구는 전략-기술-운영을 통합하는 │  │ 통제 기반 “흐름 중심 아키텍처”로 관리돼야 함 │  └────────────────────────────────────────────┘ |

요구사항 변경관리 아키텍처는 "단절 없는 연속성(End-to-End Traceability)"을 실현하기 위한 전략적 플랫폼이며, 그 핵심은 ‘요구의 생성 → 평가 → 승인 → 구현 → 검증 → 감사’에 이르는 전 과정의 흐름을 구조적으로 설계하고 통제하는 것이다.

### 아키텍처 사상

|  |
| --- |
| * 요구사항은 ****‘데이터’가 아니라 ‘관계 구조’이다****   단순한 요구 ID의 집합이 아니라, **업무 맥락, 기술 영향, 연관 시스템, 테스트 결과, 변경 이력** 등과의 복합적인 관계망으로 이해되어야 한다.  이는 \*\*구조적 모델링(Semantic Mapping)\*\*을 기반으로 한 다차원 관계 관리 체계를 요구하며, 모든 요구는 시스템 구성요소와 추적 가능해야 한다.   * ****요구**** 흐름은 ****단절 없는 아키텍처 체계를 따라야 한다****   요구사항 변경은 업무 기획자, 기술 아키텍트, 개발자, 테스터, 운영자까지 역할별 경로를 따라 이동하며 승인되고 검증되어야 한다.  따라서 아키텍처는 워크플로우 중심이 아닌, 역할 중심 흐름(Actor-Based Stream)을 가진 프로세스 통합 구조로 설계되어야 한다.   * ****변경은 ‘단일 이벤트’가 아닌 ‘파급 효과’로 설계해야 한다****   요구 변경은 단순 승인으로 끝나는 것이 아니라, 시스템 구성 요소, 테스트 항목, 릴리즈 버전, 외부 시스템, 사용자 영향까지 파급된다.  이에 따라, 아키텍처는 변경 시 \*\*계층적 영향도 분석(Impact Propagation Model)\*\*을 자동 반영하고 시각화하는 구조를 가져야 한다.   * ****모든 요구는 ‘조직 전략’과 연결되어야 한다****   요구사항은 시스템 요구가 아니라, 비즈니스 전략과 고객 요구의 변형된 결과이며, EA 구조의 상위 아키텍처 요소와 맵핑되어야 한다.  이를 위해 요구사항 ↔ 비즈니스 프로세스(BPM) ↔ 애플리케이션 ↔ 데이터 ↔ 인프라까지 연결되는 \*\*EA 통합 사상(Enterprise Binding Principle)\*\*이 적용되어야 한다.   * ****요구의 통제는 ‘자동화된 협업’ 위에서만 유효하다****   실시간 협업, 승인 워크플로우, 이력 기록, 릴리즈 연계, 알림 등은 인간이 수작업으로 관리할 수 있는 한계를 넘는다.  따라서 아키텍처는 \*\*협업 기반 + 이벤트 트리거 기반 + 권한 제어 기반의 자동 통제 구조(Event-Orchestrated Collaboration Model)\*\*로 설계되어야 한다. |

요구사항 변경관리 아키텍처의 본질은 **시스템 설계가 아닌 조직 전략의 구현 수단**이다. 툴의 기능을 나열하는 것이 아닌, **조직의 변화 흐름을 어떻게 감지하고 통제하며 확산시킬 것인지**를 설계하는 것이 바로 이 아키텍처 사상의 핵심이다.

## 요구사항 변경관리 툴의 한계점에 대한 종합적인 시사점 (정리)

### 기술적 한계 측면

요구사항 관리 도구들은 다양한 진화를 거쳐 왔지만, 여전히 실무 환경에서 다음과 같은 **기술적 제약**이 존재합니다. 이러한 한계는 도입 및 확산을 저해하고, **통합 통제 체계**를 조직 내에 안착시키는 데 걸림돌로 작용합니다.

|  |  |
| --- | --- |
| 항목 | 설명 |
| 도구 간 연계 부재 | 요구사항 도구 ↔ 형상관리(Git) ↔ 테스트도구 ↔ ALM 간 연계가 표준화되어 있지 않음. API 제공은 있으나, 실질 연동은 커스터마이징 필요 |
| 요구사항 추적 범위 제한 | 요구 ↔ 코드/테스트/릴리즈 간 연계는 일부 도구에만 존재. BPM, 데이터, 애플리케이션과의 EA 기반 맵핑 기능은 매우 제한적 |
| 모델 연계 불완전 | UML/SysML/BPMN 기반의 모델링 도구와의 연결이 불완전하거나, 수작업 변환 필요. MBSE 연계는 주로 일부 특수 툴에 국한됨 |
| 리포팅 자동화 부족 | 감사, 보안, 품질 검토 등에서 필요한 요구사항 이력, 승인 내역, 변경 사유 보고서가 표준화되지 않았거나 수작업 출력 |
| 복잡한 워크플로우 설정 | 역할별 승인 프로세스 구성이 복잡하거나, 코드 수준 스크립트에 의존해 실무 적용이 어려움 |
| 실시간 협업 기능 부재 | Jira, Jama 외 다수 도구에서 실시간 댓글, 알림, 리뷰, 변경 요청 기반 협업 인터페이스 미흡 |
| 지능형 분석 기능 미성숙 | AI 기반 요구사항 유사도 분석, 자동 분류, 변경 범위 추천 기능은 아직 상용화 초기 또는 베타 수준에 불과 |

기술적 한계점은 다음으로 정리할 수 있다.

|  |
| --- |
| * **고도화된 요구통제 체계를 원한다면, 도입 전 연계 아키텍처 설계를 먼저 고려해야 한다.** * **“툴이 기능을 제공한다”는 것과 “툴이 실무에 연계되어 작동한다”는 것은 다르다.** * **\*\*요구사항을 둘러싼 기술 생태계(테스트, 배포, 보안, 품질)\*\*와의 실질 통합이 성공의 관건이다.** |

### 요구사항 변경관리 툴 도입 시 기술 과제

요구사항 변경관리툴 도입은 단순 기술 설치가 아닌 조직 체계와의 통합 프로젝트입니다. EA 구조, 사용자 프로세스, 연계 시스템, 교육체계 등 다양한 기술과제를 함께 풀어야 실효성이 발생합니다.

|  |  |
| --- | --- |
| 항목 | 설명 |
| 전사 연계 시나리오 수립 | DevOps, ALM, CI/CD, EA, 테스트관리 시스템 등과 요구 흐름을 기준으로 사전 연계 시나리오 정립 필요 |
| 기존 도구 정리 및 통합 전략 필요 | 엑셀, 워드, Jira, 문서 서버 등으로 분산된 요구 기록 체계를 어떻게 일괄 정리하고 통합할지에 대한 전략 필요 |
| 워크플로우 설계와 승인정책 매핑 | 조직 내 승인자, 책임자, 영향 분석자 등의 R&R 구조에 따라 승인 정책과 워크플로우를 사전 설계 필요 |
| 표준 요구 메타데이터 정의 | 우선순위, 영향도, 변경 사유, 상태값 등 요구 메타데이터를 표준화하지 않으면 도구가 단순 저장소로 전락 |
| API 및 시스템 연계 개발 인력 확보 | 대부분의 고급 도구는 REST API 기반 확장을 전제로 하므로, 연계 및 운영 자동화를 위한 기술 인력 필요 |
| 도입 초반 사용자 수용성 확보 전략 | 실 사용자에게 도구 사용이 “추가 업무”로 인식되지 않도록 UI/UX 튜닝, 단계별 도입 전략 필요 |

도입 이전에 반드시 요구흐름 기준의 연계 시나리오를 명확히 설계하고, 조직내 사용자 역할 기반 워크플로우/메타정보 체계 정립이 선행되어야 한다. 툴도입은 프로젝트가 아닌 전사 체계 개선 작업의 일부로 접근 해야 된다.

### 기업환경에서 요구사항 변경관리 툴 선택 시 고려 요소

도구의 완성도 보다 더 중요한 것은 우리조직에 이 도구가 정착 가능한가입니다. 아래는 조직의 성숙, 산업특성, 인프라 조건에 따라 툴을 평가할 때 반드시 고려해야 할 요소입니다.

|  |  |
| --- | --- |
| 항목 | 설명 |
| 조직의 개발방식 | Agile 기반이면 Jira 계열, 규제 기반이면 CodeBeamer/DOORS/Polarion이 유리 |
| DevOps 생태계의 성숙도 | Jenkins, Git, Azure DevOps 등을 이미 사용 중인 조직은 연계가 쉬운 도구 선택 필요 |
| 요구사항 관리의 전사 통합 여부 | 프로젝트 단위가 아닌, 전사 요구사항 통합을 원한다면 EA 기반 연계 기능 필수 |
| 내부 감사 및 규제 대응 수준 | 감사 이력, 변경 로그, 승인 워크플로우가 필수인 조직은 Helix RM, CodeBeamer 등이 적합 |
| 전문 운영 인력 확보 여부 | 커스터마이징 및 스크립트 설정 가능한 인력의 유무에 따라 복잡한 도구 도입 여부 판단 |
| 협업 중심 조직인지 여부 | 실시간 협업, 사용자 편의 중심이라면 Jama 또는 Jira 기반 도구의 수용성이 높음 |

기술보다 중요한 건 조직의 수용력과 전략 정합성이다. 시스템 성숙도, 인프라, 사용자 문화에 따라 도구의 도입 난이도와 효과는 극단적으로 달라진다.

### 요구사항 변경관리 툴 도입 시 미래 대응을 위한 전략적 시사점

요구사항 변경관리툴의 기술은 고도화되고 있으며, AI 기반 요구 자동화, 모델 기반 연계 확대, 데이터 기반 예측 기술로 진화 중입니다. 이제는 현재의 도구를 선택하는 것이 아니라, 3~5년 후에도 대응 가능한 기반을 선택해야 합니다.

|  |  |
| --- | --- |
| 전략방향 | 설명 |
| “연결성 중심”으로 구조 재설계 | 단일 기능보다 ALM, DevOps, BPM, 데이터 구조와 통합 가능한 유연한 플랫폼 도입 필요 |
| 단계별 도입 전략 적용 | 파일럿 프로젝트 기반 도입 → 전사 확대 순으로 이행 전략 구체화 필요 |
| AI 기반 요구 분석 체계 준비 | 유사 요구 자동 분류, 변경 유사성 탐지, 중복 경고 기능 확보 가능성 고려 |
| 표준 기반 아키텍처 우선 적용 | ReqIF, OSLC, REST 기반 연계 가능한 구조를 우선 적용하여 확장성 확보 |
| 모델 연계(MBSE) 체계 준비 | DOORS, Polarion 기반 SysML, BPMN과의 연계가 가능하도록 모델 설계 체계 미리 정비 |
| 내부 역량 내재화 중심 도입 | 도구를 외부 벤더에 의존하지 않고, 내부 운영·설정 역량을 확보하는 방향으로 전략 설정 |

툴 도입은 종착점이 아니라 조직 전략 및 IT 구조 혁신의 출발점이다. 미래 변화에 유연하게 대응하려면 지금의 선택은 구조 중심 사고와 연계성 확보 전략이 기반이 되어야 한다. 기술은 계속 진화하므로, 도입 이후의 운영 내재화 전략이 중요하다.

## 기술 트렌드별 영향 요구사항 변경관리 툴 및 기술 연계 설명

요구사항 변경관리툴은 EA 모델링, 프로세스 설계, 시스템 설계, 테스트 설계 등과 유기적으로 연결 되는 것이 필수적이다. 특히 현재 기술 트렌드는 MBSE(Model Based Systems Engineering), DevOps 통합, API-First 아키텍처 등으로 이동하고 있으며, 이와 같은 기술 변화는 모델링 도구와 요구사항 관리도구의 아키텍처 통합을 요구합니다. 이에 따라 각 기술트렌드가 요구사항 변경관리툴 및 모델링 도구에 미치는 영향과 전략적 연계 포인트를 아래와 같이 정리합니다.

|  |  |  |  |
| --- | --- | --- | --- |
| 기술 트렌드 | 영향받는 모델링툴 | 연계기술 특징 | 적용 및 도입 시사점 |
| MBSE (모델 기반 시스템 설계) | Rhapsody, MagicDraw, Cameo, Visual Paradigm | SysML, UML, BPMN 등 모델 객체와 요구사항 간 양방향 연결 및 추적 | DOORS, Polarion과 통합 시 시스템 설계부터 요구 변경까지 자동 반영 가능. MBSE 기반 조직에 필수 |
| BPM 기반 프로세스 통제 | ARIS, Adonis, Bizagi, iGrafx | 프로세스 모델(BPMN)과 요구사항 항목을 맵핑하여 프로세스 변화 → 요구 자동 연동 | 요구사항을 업무단 중심으로 체계화하려는 EA/PMO 조직에서 강력한 연계 전략 수립 필요 |
| EA 기반 통합 구조 설계 | Sparx EA, Enterprise Architect, ADOxx, Archi | EA 구성요소(업무, 응용, 정보, 기술)와 요구 간 정합성 확보 및 관계 시각화 | ArchiMate 기반으로 요구사항 변경이 EA View로 반영됨. TOGAF/DoDAF 기반 EA 조직의 기본 연결 구조 |
| DevOps 파이프라인 통합 | Jenkins, GitHub Actions, Azure DevOps | 요구사항 관리도구와 개발·테스트 플랫폼 간 Webhook/API 기반 자동화 흐름 | Jira, Modern Requirements는 파이프라인 중심 요구 처리에 최적화. CI/CD 조직은 연계 시나리오 설계 선행 필수 |
| API-First / Microservice Architecture | Stoplight, Swagger, Postman, AsyncAPI | API 스펙과 요구사항을 맵핑하고 테스트/문서화/버전관리를 자동화 | 요구사항이 API 계약 수준으로 관리되어야 함. API 기반 조직은 요구-서비스 매핑 체계 우선 정립 필요 |
| 디지털 트윈/디지털 스레드 | PTC ThingWorx, Siemens NX, Dassault 3DEXPERIENCE | 물리 시스템과 디지털 모델이 실시간 연동되며, 요구사항은 모델 속 제약조건으로 반영 | 제조, 설비, 항공우주, 전장부품 등 디지털 트윈 기반 기업은 요구사항이 곧 ‘제품 동작 조건’이 되므로 통합 아키텍처 필요 |
| 클라우드 네이티브 전환 | ArchiMate, AWS Cloud Modeler, Azure Diagrams | 요구사항이 인프라(서비스 메시, 서버리스, 컨테이너) 구성 요소와 직접 연결됨 | 인프라 변화에 따른 요구사항 대응이 자동화되어야 하며, IaC(Terraform 등) 기반 연계 설계 필요 |
| AI 기반 요구 분석 및 패턴 추적 | 없음 (Emerging) | AI가 요구 유사도 분석, 중복 제거, 변경 영향 범위 추천 등의 기능 수행 예정 | 현재는 Jama, Polarion이 베타 기능 제공 중. 향후 AI 기반 요구 사전 분석 기능이 도입될 것으로 전망됨 |

기술 트렌드가 변화함에 따라, **요구사항은 더 이상 문서화된 명세 수준이 아니라 모델, 프로세스, 서비스, 인프라와 실시간으로 연동되는 구조화된 정보 객체**가 되어가고 있습니다. 이에 따라, 요구사항 관리도구는 **단독 도구가 아닌 기술 아키텍처 내 중심 흐름을 연계하는 허브 시스템**으로 설계되어야 하며, 도구 선택 시 **단기 기능이 아니라 장기 기술 트렌드와의 정합성 중심으로 평가되어야 합니다.**

|  |
| --- |
| * **조직의 기술 구조 변화는 곧 요구사항 관리 아키텍처 구조 변경으로 이어진다** * **모델링, API, DevOps, EA 구조 등과의 연계 설계는 툴 도입보다 먼저 진행돼야 한다** * **미래지향적 요구사항 관리체계는 요구 = 코드 = 모델 = 테스트라는 구조로 진화해야 한다** * **툴을 고립형으로 운용하면 기술 변화 속도에 대응하지 못하며, EA 기반 연계 시나리오 수립이 필수** * **향후 AI, 디지털트윈, 클라우드 네이티브 전환에 대비해 요구사항 관리체계를 플랫폼화하는 전략이 필요하다** |

# 요구사항 정의

본 문서는 전사 EA 표준 사업을 위한 솔루션 표준 프로파일을 도식화하고 상세에 대해 기술한 문서이다.

## 요구사항 변경관리 툴 선택 요구사항 정의표

요구사항 변경관리툴 도입 시 가장 중요한 것은 단순한 기능 나열이 아닌, 조직의 전략·업무 흐름·기술 인프라와의 정합성을 고려한 “요구항목 체계 정의”입니다. 본 항목은 실제 **전사 시스템 통합·개발환경·운영관리·보안통제** 측면에서 도구가 충족해야 할 **핵심 요구사항을 기능/비기능/전사 연계 측면**으로 구분하여 정리합니다. 이는 추후 **도구 선정 평가 기준, RFP 작성, POC(Proof of Concept) 항목**에도 바로 활용 가능합니다.

|  |  |  |
| --- | --- | --- |
| 요구사항 분류 | 세부 항목 | 설명 |
| 기능 요구사항 | 요구사항 수명주기 관리 | 생성 → 검토 → 승인 → 변경 → 종료까지의 워크플로우를 구성 가능한 구조 제공 |
|  | 요구사항 추적성(Traceability Matrix) | 요구 ↔ 설계 ↔ 테스트 ↔ 결함 간 양방향 추적 매트릭스 구성 기능 |
|  | 영향도 분석 기능 | 특정 요구사항 변경 시, 관련 테스트/기능/시스템 영향도를 자동 분석 |
|  | 승인/협업 워크플로우 | 승인자 지정, 변경 알림, 실시간 리뷰 및 협업 인터페이스 지원 |
|  | 테스트·결함 시스템 연계 | QA 시스템과의 연계를 통한 요구 기반 테스트 및 검증 추적 가능성 |
|  | 이력/버전 관리 | 요구사항의 변경 이력, 비교, 버전 복원 기능 제공 |
|  | 표준 기반 연계성(ReqIF, OSLC) | 타 시스템과의 표준 포맷 기반 연동 기능 (도구 간 이식성 확보) |
| 비기능 요구사항 | 보안 및 권한 설정 | 사용자/역할 기반의 접근 제어, 승인 권한, 변경 이력 감사 기능 포함 |
|  | 사용성(UI/UX) | 비전문가도 사용할 수 있는 직관적인 UI 제공, 요구 서식 커스터마이징 지원 |
|  | 성능 및 안정성 | 대량 요구사항 처리, 다중 사용자 동시 접근 시 응답시간 기준 확보 |
|  | 감사 및 보고서 기능 | 감사 로그, 승인 이력, 변경 이유 등 보고서 자동 생성 기능 제공 |
| 전사 연계요구사항 | EA 구성요소 연계 | 요구 ↔ 업무프로세스(BPMN), 시스템 구성도, 데이터 모델과의 연동 |
|  | DevOps 연계성 | Jenkins, Git, Azure DevOps 등 개발·릴리즈 파이프라인과의 실시간 연계 |
|  | 모델링 도구 연계 | SysML, BPMN, UML 기반 모델링 도구와의 양방향 연동 및 객체 맵핑 |
|  | API 연계성 및 확장성 | REST API, Webhook 기반 외부 연동 가능성 및 시스템 확장 구조 제공 |
|  | SaaS/On-Premise 유연성 | 조직 특성에 따라 SaaS 또는 내부 구축형 모두 지원하는 하이브리드 구조 지원 |
|  | 사용자 및 조직 단위 구성 | 부서·팀 단위로 워크스페이스 설정 및 관리 가능, 사용자 이력 관리 기능 포함 |

도구 선택 시 **단순히 기능이 있는가**만을 판단하는 것이 아니라, 우리 조직의 시스템 구조, 기술 연계 흐름, 사용자 운영 모델과 “어떻게 접합되는가”를 중심으로 요구사항을 설계해야 합니다.

이 요구사항 정의는 추후 도구를 평가하거나 외부 벤더와의 협의 시 **전략적 기준점이자 계약 요건**으로 기능하게 됩니다.

|  |
| --- |
| * 요구사항은 기술적 평가가 아닌 전략적 접합성 평가 기준이 되어야 한다 * 조직의 EA 구조와 DevOps 흐름을 고려한 요구 정의가 실질 도입 성공률을 좌우한다 * 비기능요구(보안, UI, 감사 등)는 전사 도구 통합에 핵심적인 통제 지점이다 * 도구가 가진 기능보다 “조직의 통제 구조와 얼마나 연결되는지”가 핵심이다 * 요구사항 분류는 RFP, POC, 최종 계약 조건의 기준점이므로, 전략적으로 구조화해야 한다 |

## 자가진단 체크

요구사항 변경관리툴은 단순히 기능이 많은 툴을 도입한다고 해서 성과를 보장하지 않습니다. **실제 운영 주체의 준비 수준, 프로세스 성숙도, 기술 구조의 연계성, 사용자 수용 가능성, 전략과의 정합성**이 미비할 경우, 도구는 오히려 \*\*“기능만 있는 실패 시스템”\*\*으로 전락할 수 있습니다.

따라서 아래 체크리스트는 툴 선정 전 조직이 반드시 수행해야 할 **사전 내재화 점검 항목**이며, 이는 추후 **4.3 도구 추천 전략**의 근거로도 활용됩니다.

|  |  |  |
| --- | --- | --- |
| 구분 | 점검 항목 | 체크 |
| 1. 전략·조직 관점 | 전사 수준의 요구사항 변경관리 필요성이 경영층 차원에서 공식화되었는가? |  |
|  | 요구사항을 EA 관점(업무, 데이터, 시스템, 인프라)과 연계 관리할 전략이 수립되어 있는가? |  |
|  | 요구사항 통제가 조직 내 품질/감사/보안 체계와 연계되는 통제수단으로 인식되고 있는가? |  |
| 2. 프로세스 관점 | 요구사항 생성 → 승인 → 변경 → 검증에 이르는 일관된 수명주기 프로세스가 정의되어 있는가? |  |
|  | 역할별 책임자(요청자, 승인자, 영향분석자, 테스트 검증자)가 정의되어 있는가? |  |
|  | 요구사항 메타데이터(우선순위, 상태, 변경사유, 영향도 등)가 조직적으로 정의되어 있는가? |  |
| 3. 기술 연계성 관점 | DevOps(CI/CD) 환경 또는 Git, Jenkins, Azure DevOps 등의 도구를 이미 활용하고 있는가? |  |
|  | 조직 내 모델링 도구(BPMN, UML, SysML)와 연계 가능한 구조가 사전 정의되어 있는가? |  |
|  | 테스트관리도구(TestRail, Xray 등) 또는 결함관리도구와 요구 간 연계 시나리오가 존재하는가? |  |
| 4. 시스템·운영 관점 | 도구 운영 전담 인력(관리자, 기술 설정자, 연계 개발자 등)이 확보되어 있는가? |  |
|  | 보안, 감사, 변경로그 기준 등 통제 정책이 존재하며, 시스템에 반영될 수 있는가? |  |
|  | SaaS와 On-Premise 중 조직에 적합한 인프라 운영 방식을 내부적으로 결정했는가? |  |
| 5. 사용자 수용성 관점 | 도구 도입에 대해 실사용자(요구 담당자, PM, 테스터 등)의 반응과 수용성 진단이 이루어졌는가? |  |
|  | 과거 실패 사례 또는 툴 도입에 대한 저항 요인이 파악되어 있고, 대응 전략이 수립되어 있는가? |  |
|  | 도입 이후 단계적 정착 로드맵(파일럿 → 확산 → 내재화)이 설계되어 있는가? |  |

이 체크리스트는 단순히 준비 정도를 측정하는 것이 아니라, **툴을 통해 달성하려는 조직 전략이 실제로 실현 가능한지, 그에 필요한 체계와 인프라가 준비되어 있는지를 점검하는 전략 툴입니다.**

|  |
| --- |
| * 도구 선정은 준비되지 않은 조직에는 “부채”가 될 수 있다 * 단순히 도입하는 것이 아니라, 변화·연계·운영·수용력 전반을 아우른 내재화 전략이 병행되어야 한다 * 실질 도입 성공률은 기능보다 사전 준비 상태에 좌우된다 * EA 기반의 요구 흐름, 역할 기반 워크플로우, 연계 시스템 설계가 준비되지 않으면 도구는 무용지물이다 * 체크리스트는 내부 승인, 투자 의사결정, 전략 정합성 평가에도 그대로 활용 가능하다 |

## 자가진단 기반 선택 가이드

조직의 도구 도입은 단순한 기능적 적합성뿐 아니라 **전사 전략과의 정합성, 내부 운영 준비 수준, 기술 연계성, 수용성**에 따라 효과성이 결정됩니다. 따라서 도구를 선택할 때는 사전 자가진단을 통해 조직의 준비 성숙도를 계량화하고, **그에 맞는 도구 유형과 전략을 적용하는 방식이 가장 현실적이며 실행 가능성이 높습니다.**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 자가진단 점수 구간 | 조직 상태 진단 | | 추천 도구 유형 | 대표 툴 예시 | | 도입 전략 요약 |
| 90 ~ 100점 | **고성숙 EA 조직**  전사 통합 통제 및 시스템 기반 운영 정착 | | 통합형 요구사항 + ALM + 모델링 연계형 도구 | **CodeBeamer**, **Polarion**, **DOORS Next** | | 전사 기반 연계 체계 수립, MBSE/DevOps/QA 자동화 통합 설계로 확산 전략 수립 |
| 75 ~ 89점 | **DevOps 기반 실무 조직**  부분 연계 도입 + 정착 가능 | | 요구-테스트-릴리즈 연계형 / 협업 중심 도구 | **Jama Connect**, **Jira (Xray/Structure)**, **Helix RM** | | 요구-테스트 연계 + 릴리즈 통제 중심으로 점진적 확대. 사용자 중심 운영 전략 필수 |
| 60 ~ 74점 | **기초 프로세스 확립 조직**  요구 흐름은 있으나 자동화 부족 | | 경량화된 요구관리 중심 도구 + 협업 기능 위주 | **Jira (Structure Only)**, **Modern Requirements**, **ReqView** | | 핵심 업무부터 시작하여 단계별 도입 추진, 상위 아키텍처 연계는 2단계에서 설계 |
| 40 ~ 59점 | **개별 프로젝트 중심 조직**  산발적 요구관리, 정형화 미비 | | Wiki/Excel 대체 도구 + 협업 및 이력 중심 | **Confluence + Jira Lite**, **ReqIF.ac**, **SpiraTeam** | | 요구 메타데이터 정리, 역할 기반 승인 시나리오 설계부터 선행 필요 |
| 0 ~ 39점 | **요구사항 관리 기반 미정착 조직**  통제 체계 부재, 수작업 중심 | | 도구 도입보다 조직 통제 역량 확보 선행 필요 | 도구 도입 연기 권고 | | 도입 전 워크플로우 정의, 프로세스 교육, R&R 설정 후 파일럿 기반 평가 추진 권고 |
| 점수 구간 | | **조직이 먼저 확보해야 할 선결 사항** | | | **적용 전략 요약** | |
| 90~100점 | | 요구 흐름의 코드-테스트-릴리즈 일원화 시나리오, 감사 기준 자동화 설계 | | | 자동화 우선 전략, API 기반 통합 플랫폼 도입 및 사전 벤더 교육 병행 | |
| 75~89점 | | 모델 연계 또는 테스트 툴 통합 전략서, 사용자 중심 도입 시나리오 | | | 실무 주도 도입 후 EA/QA 체계 연계로 확대 전략 수립 | |
| 60~74점 | | 요구사항 메타데이터 표준 정의, 승인 정책 체계화 | | | 경량 도구 + 요구 흐름 정형화 병행, 운영자 교육 필수 | |
| 40~59점 | | 역할 기반 승인 체계, 요구 수명주기 정의 | | | 파일럿 중심 적용, 기존 문서 → 요구 시스템 이관 프로세스 설계 우선 | |
| 0~39점 | | 전체 요구 흐름 정의, 실사용자 인터뷰 및 공감대 확보 | | | 툴 선정 보류. 조직 내역 정비 및 요구 흐름 체계화 프로젝트 우선 수행 | |

요구사항 변경관리툴의 도입은 **조직의 전략적 방향성과 기술 내재화 역량**을 기반으로 체계적으로 접근해야 합니다. 자가진단 점수는 도구 선정의 출발점이며, **실제 도입 성공률을 높이기 위한 리스크 기반 전략적 기준점**으로 반드시 활용해야 합니다.

|  |
| --- |
| * 툴 선택은 “현재 조직의 역량”과 “미래 변화 방향”의 교차지점에서 판단해야 한다 * 성숙도가 낮은 조직은 도구 도입보다 “체계 정비와 사용자 기반 마련”이 선행되어야 한다 * 중간 수준의 조직은 기능 중심 도구보단 협업/통제 중심 도구가 정착에 효과적이다 * 고성숙 조직은 도구보다는 아키텍처 중심의 통합 시나리오 설계가 핵심 전략이다 * 자가진단은 툴 도입뿐 아니라 EA 통제체계 확산을 위한 내부 전략 수립 도구로 활용 가능하다 |

# 도구 선정 가이드

본 문서는 전사 EA 표준 사업을 위한 솔루션 표준 프로파일을 도식화하고 상세에 대해 기술한 문서이다.

요구사항 변경관리툴을 선택할 때 가장 중요한 것은 **단순한 기능 존재 여부가 아니라**, 그 도구가 조직의 **EA 구조, DevOps 체계, 운영 전략, 품질 통제 정책**과 **얼마나 정합성 있게 접합될 수 있는가**입니다.

본 항목에서는 도구 선정 시 반드시 고려해야 할 **전사 전략 관점의 핵심 조건**들을 **기능/기술/운영/정책/전략** 5개 축으로 구분하여 제시합니다. 이는 RFP 조건, 기술심의 기준, 선정평가 항목 정의 시 필수 반영되어야 할 전략 기준입니다.

## 선정조건 정리

|  |  |  |
| --- | --- | --- |
| 구분 | 선정 조건 항목 | 설명 |
| 1. 기능 정합성 | 수명주기 기반 요구 흐름 지원 | 요구의 생성 → 분석 → 승인 → 변경 → 검증 → 종료까지 전 주기 관리 가능 여부 |
|  | 요구-테스트-릴리즈 추적성 보장 | 요구사항이 테스트, 결함, 릴리즈와 직접 연결되고 추적 가능한 구조 보유 여부 |
|  | 협업 기반 리뷰 및 승인 기능 | 실시간 의견 교환, 댓글, 알림, 승인 이력 등 사용자 협업 중심 구조 보유 여부 |
| 2. 기술 통합성 | DevOps/CI-CD 연계 인터페이스 | Git, Jenkins, Azure DevOps 등과의 API 연동 및 자동화된 워크플로우 지원 여부 |
|  | EA/모델링 도구 연계 가능성 | BPMN, UML, SysML 등의 모델링 도구와 객체 기반 매핑 및 동기화 지원 여부 |
|  | 표준 API 및 인터페이스 제공 | REST API, Webhook 등 다양한 시스템 연계 구현 가능한 표준 기반 구조 여부 |
| 3. 운영 편의성 및 보안 | 사용자 중심 UI/UX | 도입 후 실무자들이 자연스럽게 사용할 수 있는 UX 구성 및 대시보드 제공 여부 |
|  | 역할 기반 권한 관리 | 역할별 읽기/쓰기/승인/변경 이력 권한 설정 및 보안 감사 기록 구조 제공 여부 |
|  | SaaS/On-Premise 선택 가능성 | 조직의 인프라 정책에 따라 유연한 구축/운영 방식 선택 가능 여부 |
| 4. 정책 대응력 | 감사/보안/품질 정책 수용 | 변경 이력, 승인 내역, 사유 기록 등이 외부 감사, 내부 규정 대응에 활용 가능한가 |
|  | 요구 메타데이터 정책 설정 가능 | 우선순위, 중요도, 출처, 영향도 등 요구 메타 필드 정의 및 분석 가능 여부 |
|  | 규제 기반 업종 지원 가능 | ISO26262, FDA, ASPICE 등 산업 규제 환경에 적합한 통제 구조 제공 여부 |
| 5. 전략 정합성 및 확장성 | EA 체계와의 구조적 연계 가능성 | EA 요소(업무/데이터/응용/기술)와의 연계 구조 설계가 가능한 구조인지 |
|  | 확장 가능한 연계 아키텍처 | 기존 ALM, 테스팅, 형상관리, 서비스 관리 등과 연계 가능한 구조인지 |
|  | 벤더 기술 지원 및 교육 체계 | 초기 구축 및 운영 내재화를 위한 기술지원, 사용자 교육 제공 여부 |

선정 조건은 단순 비교 항목이 아니라, **우리 조직이 구축하고자 하는 전략적 아키텍처에 이 도구가 얼마나 자연스럽게 접합될 수 있는가를 평가하는 구조적 질문지**입니다. 조직 전략, 기술 스택, 연계 체계, 내부 운영 능력을 모두 고려한 조건이 정교하게 정의되어야만 **도입 이후의 실패 가능성을 줄이고, 장기적 운영 가능성을 확보할 수 있습니다.**

|  |
| --- |
| * 기능이 아닌 구조적 정합성 기반의 조건 정의가 툴 선정의 첫 출발점이다 * 조직이 목표로 하는 요구통제 체계의 수준에 따라 조건 우선순위는 달라질 수 있다 * 도구의 성공 여부는 도입 이전의 조건 정의의 수준에 따라 결정된다 * RFP/POC/벤더 검증의 기준점은 기능이 아닌 “아키텍처 접합력”이어야 한다 * 선정 조건은 곧 “도입 후 통제 가능성”과 “전사 적용의 지속성”을 보장하는 장치다 |

## 요구사항 변경관리 툴 선정을 위한 평가기준 및 가중치 구성

요구사항 변경관리툴의 선정은 기능적 만족도뿐 아니라, **기술 연계성, 전략 정합성, 운영 현실성, 확장 가능성**까지 고려한 정량적 평가가 필요합니다. 본 평가 기준은 **정책 수립 조직, EA 부서, 정보화 기획부서, 품질/운영 통제 담당자** 등 복수의 이해관계자들이 합리적으로 도구를 비교할 수 있도록, **전략 가중치 기반 구조**로 설계되었습니다.

|  |  |  |
| --- | --- | --- |
| 평가 항목 | 가중치 (%) | 세부 평가 내용 |
| 1. 기능적 완성도 | **25%** | 수명주기 관리, 요구-테스트-릴리즈 간 추적성, 승인 프로세스, 이력/버전 관리, 협업 기능 등 기능의 내재화 수준 |
| 2. 기술 연계성 | **20%** | DevOps, Git, Jenkins, Azure DevOps 등과의 연동 구조, REST API, Webhook, ReqIF/OSLC 연계 지원 여부 |
| 3. 운영 효율성 | **15%** | 사용성(UI/UX), 사용자 역할관리, 템플릿 활용성, 관리자 기능, SaaS/On-Prem 구성 유연성 등 |
| 4. 전략 정합성 | **15%** | 조직의 EA 구조(업무, 시스템, 데이터)와 정합성, BPM/UML 모델 연계 가능성, R&R 기반 승인체계 설정 가능 여부 |
| 5. 감사·보안 대응력 | **10%** | 감사 로그, 변경 사유 저장, 승인 이력 보관, 역할 기반 보안 설정, 규제 대응 지원(FDA, ISO 등) 여부 |
| 6. 벤더 신뢰도 및 기술지원 체계 | **10%** | 초기 구축 지원, 유지보수 전략, 기술 문서화 수준, 교육 자료 제공, 커스터마이징 유연성 |
| 7. 확장성 및 지속가능성 | **5%** | 도구의 커뮤니티 활성화, 제품 로드맵, SaaS 클라우드 확장성, 다국어/글로벌 대응 여부 등 장기 운영 요소 |
| 항목 | **도구 비교 시 적용 예** | |
| 기능적 완성도 | DOORS는 요구-테스트-모델 연계가 강점 / Jira는 협업성과 UI/UX 우수 | |
| 기술 연계성 | Jira + Xray는 DevOps 친화적 / Polarion은 ReqIF 기반 연계에 우수 | |
| 운영 효율성 | Jama는 사용자 중심 리뷰 최적화 / CodeBeamer는 설정 복잡도가 있음 | |
| 전략 정합성 | EA 기반 체계 가진 조직은 DOORS·Polarion 유리 / 단일팀 기반은 Jira 유리 | |
| 감사·보안 대응 | CodeBeamer, DOORS는 인증체계 강점 / Jira는 일반 감사기능은 보완 필요 | |
| 벤더 신뢰도 | IBM, Siemens는 안정성 높음 / 신생 SaaS형 도구는 기술 의존도 주의 필요 | |
| 확장성 | Jira, Jama는 API 확장 자유도 높음 / 일부 온프레미스 도구는 제약 있음 | |

툴의 성능과 적합성을 객관적으로 비교하기 위해서는 **기능 중심이 아닌, 전략 연계·기술 정합·운영 통제·확장성 중심의 구조화된 평가 기준이 필수적**입니다. 평가항목의 가중치는 조직의 전략 목표, 인프라 수준, 통제 요구사항에 따라 **탄력적 조정이 가능하도록 설계되어야 하며**, **도입 실패율을 줄이고 전사 확산 가능성을 확보하는 전략적 기반 도구**로 기능해야 합니다.

|  |
| --- |
| * 툴 선정의 실패는 기능 미비보다 평가 기준 부재에서 발생한다. * EA, DevOps, 품질, 보안 등 다각도의 전략 연계성이 평가 기준에 반영되어야 한다. * 전사 도입일수록 단일 기능이 아닌 통제 체계/연계성/운영성 중심으로 도구를 평가해야 한다. * 가중치 구조는 조직 우선순위(예: DevOps 중심 vs 품질중심 등)에 따라 유연하게 재설계될 수 있어야 한다. * 전략적 기준 기반 도구 선정은 투자 효율성과 도입 후 정착률 향상의 핵심 요소이다. |
|
|
|
|

## 요구사항 변경관리 툴 평가항목 점수표

요구사항 변경관리툴을 도입할 때는 **조직의 목표와 기술 인프라, 수용 전략에 정합되는지 여부를 정량적으로 판단**할 필요가 있습니다.  
본 점수표는 전략적 평가기준을 기반으로 대표적인 도구들을 다음 항목으로 비교합니다:

* IBM DOORS / DOORS Next
* Jira + Xray / Structure
* Jama Connect
* CodeBeamer
* Polarion

각 항목은 10점 만점 기준 상대점수로 산정되며, 5.2 항목의 가중치를 적용해 **총점 100점 기준으로 최종 비교 분석**합니다.

### 평가항목 점수표

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 평가 항목 | 가중치(%) | DOORS Next | Jira + Xray | Jama Connect | CodeBeamer | Polarion |
| 기능적 완성도 | 25% | 9.5 (23.75) | 7.5 (18.75) | 8.8 (22.00) | 9.0 (22.50) | 9.2 (23.00) |
| 기술 연계성 | 20% | 8.5 (17.00) | 9.5 (19.00) | 8.2 (16.40) | 8.7 (17.40) | 9.0 (18.00) |
| 운영 효율성 | 15% | 6.8 (10.20) | 9.4 (14.10) | 9.2 (13.80) | 7.5 (11.25) | 7.8 (11.70) |
| 전략 정합성 | 15% | 9.2 (13.80) | 7.2 (10.80) | 8.5 (12.75) | 9.0 (13.50) | 9.1 (13.65) |
| 감사·보안 대응력 | 10% | 9.4 (9.40) | 6.5 (6.50) | 7.8 (7.80) | 9.3 (9.30) | 9.5 (9.50) |
| 벤더 신뢰도 및 기술지원 | 10% | 9.5 (9.50) | 8.5 (8.50) | 8.8 (8.80) | 8.0 (8.00) | 9.2 (9.20) |
| 확장성 및 지속가능성 | 5% | 8.0 (4.00) | 9.5 (4.75) | 9.2 (4.60) | 8.3 (4.15) | 8.7 (4.35) |
| 총점 (100점 기준) | - | **87.65** | **82.40** | **85.15** | **86.10** | **89.40** |

표를 통해 알 수 있듯이 \*\*Swagger(OpenAPI)\*\*가 문서 자동화, 표준 지원, 유지보수 관점에서 가장 우수한 평가를 받았습니다. 그 뒤를 **Postman**과 **Redoc**이 따르며, 협업과 사용자 경험 측면에서 강점을 보였습니다. 반면, **Spring REST Docs**와 **Slate**는 기능 커버리지와 자동화, 협업성 측면에서 점수가 낮았습니다.

### 도구별 적용 전략 해석

|  |  |
| --- | --- |
| 도구 | 전략적 분석 |
| DOORS Next | 전통적 엔터프라이즈 통제와 MBSE 정합성이 뛰어나며, 규제 대응 및 감사 통제 필요 조직에 적합. 단, UI 및 사용자 편의성은 약점. |
| Jira + Xray | DevOps 중심의 개발 조직에 매우 적합. 협업성과 유연성이 뛰어나고 운영 효율성이 강점이나, 전략 연계 및 감사 측면은 보완 필요. |
| Jama Connect | 사용자 중심 협업 구조와 실시간 리뷰 강점. 규제 산업 대응 가능하며, 중견~대기업 확산에 적합. 기술 연계는 구조 설계 필요. |
| CodeBeamer | 테스트-릴리즈-품질 연계에 특화되어 있어 ASPICE/FDA 대응 조직에 적합. 초기 설정 복잡도 및 UI는 보완 요소. |
| Polarion | 전사 EA 정합성, 모델 연계, 보안 감사 대응 측면에서 최고 수준. 다만, 사용자 편의성 및 초기 학습 비용은 고려 필요. |

이 점수표는 단순한 기능 비교를 넘어서 **조직의 전략 방향성과 시스템 구조, 운영 전략, 기술 아키텍처 연계성까지 고려한 도구 선택 기준**입니다.

단일 점수보다는 \*\*조직의 우선순위(예: DevOps, 품질 통제, EA 연계)\*\*에 따라 가중치 항목을 조정하거나 부분 점수 기준을 재설계하는 방식으로 **전략적 커스터마이징**이 가능합니다.

|  |
| --- |
| * 모든 도구는 전략적 목적과 조직 구조에 따라 적합성과 가치를 달리한다 * 단일 기능 비교보다 전략 가중치 기반 분석이 도입 성공률을 좌우한다 * EA, 품질, 보안, DevOps 중 어떤 전략을 우선할 것인지에 따라 평가 방식은 유동적으로 설계되어야 한다 * 점수 결과는 최종 선정이 아니라 전사 전략 방향성과의 접합 시뮬레이션 도구로 사용해야 한다 |

## 프로젝트 유형별 · 상황별 · 규모별 맞춤 선정표

### 프로젝트 유형별/상황별 추천표

요구사항 변경관리툴은 “모든 조직에 가장 좋은 도구”가 아니라, “우리 조직의 전략, 프로젝트 특성, 기술 운영 방식에 가장 정합성 높은 도구”를 선택해야 성공합니다.

본 항목은 **전사 EA 관점에서 프로젝트 유형을 5가지 전략군으로 분류**하고, 각 유형에 적합한 도구와 그 선정 이유를 전략적으로 매핑한 표입니다.

|  |  |  |  |
| --- | --- | --- | --- |
| 프로젝트 유형 | 상황 설명 | 추천 툴 | 선정 이유 (전략적 적합성) |
| ① 전사 EA 기반 통합 시스템 구축형 | * 전사 BPM, APP, DATA, INFRA 구조와 정합성 필요. * 요구-테스트-릴리즈-감사 전 흐름 통제 필요 | **Polarion**, **DOORS Next**, **CodeBeamer** | EA 구조 연계성 우수, 모델 기반 연계(ArchiMate, SysML), 감사 대응 기능 내장. 전사 통제 기반 운영 적합 |
| ② DevOps 기반 마이크로서비스 프로젝트 | * 릴리즈 속도 우선, 요구사항이 이슈 기반으로 빠르게 반영. CI/CD, Git, Jenkins 중심 구조 | **Jira + Xray**, **Modern Requirements** | Git/Jenkins 연계 최적화, 협업성과 실시간 연계, 릴리즈 중심 트레이서빌리티 우수. 스크럼/칸반 운영 친화적 |
| ③ 규제/품질중심 산업형 (자동차, 의료, 항공) | * ISO26262, FDA, DO-178C 등 외부 규제 준수 필수. * 요구사항-테스트-품질 완전 통합 요구 | **CodeBeamer**, **Helix RM**, **DOORS** | 인증 구조 내장, 테스트 추적성 보장, 변경 로그 보존 강력. 문서 기반 검토, 릴리즈 증적 관리 최적화 |
| ④ 중견기업/복합 사업 다수 병행형 | * 요구-기획-운영까지 일괄 추적 필요. * PM, BA, QA 간 협업이 핵심 | **Jama Connect**, **Jira (Structure)** | 사용자 중심 협업 기능, 실시간 리뷰, 직관적인 트레이서빌리티 UI. 애자일·협업 혼합 구조 대응 |
| ⑤ 요구사항 기반 컨설팅/솔루션 개발형 | * 고객 맞춤형 요구사항 분석·관리 필요. * 사전 정형화된 요구정의, 문서 중심 구조 | **ReqIF.ac**, **ReqView**, **DOORS** | 문서 기반 요구 관리, 버전관리 및 승인 추적 명확, 표준 기반 요구 템플릿 지원 |
| 프로젝트유형 | * 상황설명 | 추천툴 | 선정이유 |
| 공공기관 프로젝트 | * 표준 준수 필수, 문서 이력 관리와 정적 문서화 중요 | **Spring REST Docs** | 정적 문서 기반, 테스트 기반 문서화로 감사 및 기록 보존에 유리 |
| 스타트업 MVP 개발 | * 빠른 개발, 빠른 배포, 쉬운 공유 필요 | **Swagger UI** | 문서 자동 생성, 테스트 가능한 UI 제공, 빠른 피드백 루프 가능 |
| 대기업 내부 API 게이트웨이 구축 | * 다양한 팀과 협업, 명확한 계약 기반 필요 | **OpenAPI + Redoc** | 명세 기반 문서화(OpenAPI) + 고급 뷰 UI 지원(Redoc), 협업 효율성 탁월 |
| 클라우드 기반 MSA 프로젝트 | * 마이크로서비스 간 연동 문서 자동화, 변경 이력 관리 | **Swagger (OpenAPI)** | REST API 표준을 기반으로 다양한 마이크로서비스와 연동 용이 |
| 외부 개발자 대상 공개 API | * 개발자 포털 구축, 문서 UI 디자인/사용성 중요 | **Slate** | Markdown 기반 정적 사이트 생성, 예제 기반 명확한 가이드 제공 |
| 테스트 코드 기반 설계 적용 | * 테스트 기반 개발(TDD)과 연계되는 문서화 필요 | **Spring REST Docs** | 테스트 코드 기반 자동 문서 생성 가능, 코드 품질 및 문서 품질 동시 확보 가능 |

도구 선택은 조직이 운영하는 **업무 흐름, 통제 수준, 릴리즈 정책, 기술 연계성**과 맞아야 정착됩니다. “유명한 도구”가 아니라 \*\*“우리 업무/기술 환경과 가장 전략적으로 결합 가능한 도구”\*\*가 정답입니다.

이 유형 매핑은 조직의 투자 대비 성공 확률을 높이는 가장 현실적이고 전략적인 접근 방식입니다.

|  |
| --- |
| * 도구의 기능보다 중요한 것은 “적합한 전략 유형에 맞는 접합성”이다 * EA 기반, DevOps 기반, 규제 기반, 협업 기반 등 전략군별 매핑은 정착률과 ROI를 결정한다 * 전사 도입이라면 하나의 도구가 모든 프로젝트를 커버할 수는 없다는 현실 인식이 필요하다 * 유형별로 도구를 구분하여 사용하거나, 전사 표준+보완 도구로 병행 운영 전략을 검토할 수 있다 * 도입 전에 우리 조직의 요구 흐름/품질 흐름/기술 연계 흐름을 먼저 시각화한 후 도구를 매핑하는 것이 최적 전략이다 |

### 프로젝트 규모별 적용 가이드

요구사항 변경관리툴은 **프로젝트의 규모, 구조 복잡도, 관리 통제 범위**에 따라 **도구의 기능/확장성/운영 난이도/정착률**이 크게 달라집니다.

따라서 단순히 기업 크기에 따른 구분이 아니라,  
\*\*“관리해야 할 요구사항의 양, 복잡도, 연계 범위, 품질 감사 수준”\*\*을 기준으로 도구를 매핑하는 것이 가장 현실적이며 효과적입니다.

|  |  |  |  |
| --- | --- | --- | --- |
| 프로젝트 규모 | 기준 설명 | 추천 도구 | 선정 이유 (EA 전략 관점) |
| 1. 초대형 전사/국가사업 (EA/통합/공공) | * 요구사항 1,000건 이상, 수명주기 12개월 이상. * EA/BPM/데이터/시스템 통합 기반 필수 | **DOORS Next**, **Polarion**, **CodeBeamer** | EA 구조 연결, 모델 연계, 다부서 협업 및 통제, 릴리즈-품질-감사 대응. 전사 통합 통제에 최적화 |
| 2. 대규모 시스템 구축형 (복수 모듈, MSA 포함) | * 요구사항 500~1,000건, 모듈/팀 분산 구조. * DevOps, ALM, 테스트 연계 필요 | **Jama Connect**, **CodeBeamer**, **Jira + Xray** | 요구 흐름 분산, 테스트/이슈 기반 연계, 승인 통제, 시각화된 트레이서빌리티 |
| 3. 중규모 솔루션 구축형 (10~30명 개발팀) | * 요구사항 100~500건, 단일 시스템 + 기능별 복잡도 존재. * QA 및 고객 커뮤니케이션 중요 | **Jira (Structure)**, **Jama**, **Modern Requirements** | 사용자 중심 협업, 테스트 연계, 변경 관리 구조. 기획~릴리즈 흐름 정형화 가능 |
| 4. 소규모 애자일 프로젝트 (스타트업/파일럿형) | * 요구사항 30~100건, 빠른 피드백 중심. * 제품/서비스 릴리즈 반복 우선 | **Jira Core**, **ReqView**, **SpiraTeam** | 간결한 기능 + 빠른 도입 + UX 최적화. 애자일 기반 구조에 내재화 용이 |
| 5. 문서 기반 경량 프로젝트 (SI 컨설팅형) | * 요구사항 30건 이하, 문서 중심. * 계약 기반, 공공 프로젝트 등 | **ReqIF.ac**, **DOORS Classic**, **Excel + 관리도구** | 정형 템플릿 + 승인 흐름 + 버전 관리 필요. 문서 → 시스템 이관 시 용이 |

프로젝트 규모에 따라 도구가 가져야 할 **기능 범위, 연계 구조, 도입 전략, 조직 수용성**은 다릅니다.  
기능이 많은 도구를 쓰는 것이 능사가 아니라, **우리 조직의 통제 역량과 프로젝트 요구 흐름에 맞춘 선택이** 가장 성공 가능성을 높입니다.

또한, 대규모 프로젝트일수록 **파일럿 도입 → 구조 설계 → 단계별 확산 전략**을 함께 세워야 합니다.

|  |
| --- |
| * 도구는 규모에 맞춰 “기능 + 연계 + 운영 내재화 가능성” 기준으로 선택해야 한다 * 대규모는 통합 구조 기반, 중소규모는 협업 기반 구조로 분리하여 접근해야 한다 * 한 프로젝트에 여러 툴이 연동될 수도 있음을 고려해 “툴 간 연계 전략”도 함께 수립해야 한다 * 전사 확산형 도입이라면 사전 정비, 기술적 연계 설계, 이해관계자 합의 구조를 반드시 선행해야 한다 * 규모가 작더라도 요구 흐름이 명확하고 통제 체계가 정립된 조직이라면 고급 도구도 효과적으로 운영 가능하다 |

# 부록

## 이해관계자 분석

|  |  |  |  |
| --- | --- | --- | --- |
| 이해관계자 그룹 | 주요 요구사항 및 관심사 | 영향력 | 지원 필요성 |
| 전사 전략기획팀 | EA 구조 내 통제 체계 정립, 전사 프로세스-시스템 간 정합성 확보 | 높음 | 정책 정합성 유지 및 도입 논리 강화 지원 필요 |
| IT기획/품질관리팀 | 변경 이력 감사 대응, 테스트·품질 추적성 확보, 감사 레포트 자동화 | 매우 높음 | 기능 요구 우선반영, 테스트관리 연계 설계 지원 |
| 개발 조직 (Dev 팀) | Jira/DevOps 연계, 이슈 중심 요구 처리, 배포 트레이서빌리티 | 중간~높음 | 인터페이스/API 구성, 협업 UX 최적화 필요 |
| 테스트/QA 부서 | 요구사항 기반 테스트케이스 연계, 결함 분석과 요구 추적 | 중간 | 요구-테스트 통합 구조 설계 지원 필요 |
| 정보보안팀 / 감사팀 | 승인/변경 이력 자동 저장, 감사 대응 레포트, 권한 기반 통제 | 낮음~중간 | 정책 기반 보안 요소 도입 협의 필요 |
| 프로젝트 관리자 (PM) | 요구 수명주기 가시화, 승인체계 통제, 일정 지연 원인 추적 | 중간 | 대시보드 제공, 승인 워크플로우 최적화 |
| 최종 사용자 (업무팀) | 직관적 UI/UX, 요구 등록·검토 용이성, 협업성 | 중간 | 사용자 교육, 템플릿 구성, UI 설정 지원 |
| 벤더/외주 개발사 | 요구 공유 체계, 변경 통보, 형상 연계 | 낮음 | 제한적 접근 권한 구조 및 워크스페이스 설정 |

## 요구사항

### 기능요구사항

#### 필수기능

|  |  |  |  |
| --- | --- | --- | --- |
| 영역 | 기능 요구사항 | 우선순위 | 중요도 |
| 요구 수명주기 관리 | 요구사항 생성-검토-승인-변경-종료 단계별 워크플로우 지원 | 1 | ★★★★★ |
|  | 상태값 기반 요구 추적 (초안, 승인됨, 변경됨 등) | 2 | ★★★★☆ |
| 요구 트레이서빌리티 | 요구 ↔ 설계 ↔ 테스트 ↔ 릴리즈 간 양방향 추적 매트릭스 구성 | 1 | ★★★★★ |
|  | 요구 ↔ 결함 간 연계성 시각화 (Impact Trace) | 2 | ★★★★☆ |
| 협업 및 승인 통제 | 역할 기반 승인 체계 (요청자/승인자/영향분석자) 설정 기능 | 1 | ★★★★★ |
|  | 변경사항 알림, 댓글, 실시간 협업 기능 내장 | 2 | ★★★★☆ |
| 변경이력 및 감사 통제 | 요구 변경 이력 자동 저장 및 비교 기능 | 1 | ★★★★★ |
|  | 승인 이력 및 변경 사유 감사 로그 생성 | 2 | ★★★★★ |
| DevOps 연계 및 API 지원 | Git/Jenkins/Azure DevOps 등과 연계 가능한 REST API 제공 | 2 | ★★★★☆ |
| 모델링 도구 연계 | SysML, UML, BPMN 기반 객체와요구사항 간 링크 설정 기능 | 2 | ★★★★☆ |
| 요구 서식 커스터마이징 | 조직 내 메타데이터(우선순위, 출처 등) 정의 및 관리 기능 | 1 | ★★★★☆ |
| 템플릿/양식 제공 | 요구 정의서, 승인서, 변경 통지서 등 기본 템플릿 제공 | 2 | ★★★☆☆ |
| 검색/필터/리포트 | 요구사항 다조건 검색, 변경 항목 필터링, PDF/CSV 리포트 출력 | 1 | ★★★★☆ |

#### 중요기능요구사항

|  |  |  |  |
| --- | --- | --- | --- |
| 영역 | 기능 요구사항 | 우선순위 | 중요도 |
| 요구분류 및 템플릿화 | 요구사항 분류 체계(기능/비기능/보안 등) 설정 및 필터링 | 1 | ★★★★☆ |
|  | 요구 정의서 템플릿 커스터마이징 및 정형 양식 저장 | 2 | ★★★★☆ |
| 재사용성 및 라이브러리 관리 | 반복 사용되는 요구사항 템플릿 저장 및 재사용 기능 | 2 | ★★★★☆ |
|  | 과거 프로젝트 요구사항 검색/활용을 위한 레포지토리화 기능 | 2 | ★★★★☆ |
| 요구사항 품질 진단 | 요구 중복, 누락, 유사성 자동 탐지 기능 | 3 | ★★★☆☆ |
|  | 요구 문장 패턴 분석 및 품질 레벨 자동 검증 (예: 모호성 감지) | 3 | ★★★☆☆ |
| 요구-비즈니스 연계 시각화 | BPMN 기반 업무 프로세스와 요구사항 간 연결 시각화 | 2 | ★★★★☆ |
| AI 기반 분석/추천 기능 *(선택 고도화)* | 유사 요구 자동 분류, 변경 파급 예측 추천 | 4 | ★★☆☆☆ |
| 이해관계자 리뷰 이력 관리 | 요구사항별 검토자, 검토의견, 이력 저장 및 필터 조회 기능 | 2 | ★★★☆☆ |
| 시각화 대시보드 구성 | 요구 생성 현황, 승인 현황, 변경 건수, 릴리즈 연계 상태 등 가시화 | 2 | ★★★★☆ |
| 규정 기반 보고서 출력 | 커스터마이징 가능한 PDF/Excel 보고서 출력 기능 | 3 | ★★★☆☆ |

### 비기능 요구사항

|  |  |  |  |
| --- | --- | --- | --- |
| 카테고리 | 요구사항 | 측정기준 | 목표치 |
| 성능 (Performance) | 다중 사용자 동시 접속 시 응답 속도 | 사용자 100명 동시 요청 시 화면 응답시간 | 2초 이내 |
|  | 데이터 저장/조회 처리 시간 | 1,000건 이상의 요구 목록 조회 시 처리 시간 | 3초 이내 |
| 가용성 (Availability) | 시스템 연중 무중단 운영 | 월 평균 가동률(SLA 기준) | 99.9% 이상 |
|  | 장애 발생 시 복구 시간 | 장애 발생 시 RTO (복구목표시간) | 1시간 이내 |
| 확장성 (Scalability) | 조직 단위 증가 시 성능 영향 최소화 | 팀/부서 수 100단위 증가 시 성능 저하율 | 5% 이내 |
|  | 사용자 수 증가 시 서버 확장 용이성 | 100명 → 1,000명 확장 시 인프라 증설 유연성 | 수평 확장 가능 구조 |
| 보안성 (Security) | 접근제어 및 권한 설정 | 역할 기반 사용자 권한 제어 체계 | 필수 |
|  | 감사 이력 및 변경 추적 | 요구/승인/변경/접속 로그 저장 및 조회 가능 | 12개월 이상 보관 |
|  | 개인정보 및 내부정보 보호 | HTTPS 통신, 내부망 적용, 암호화 저장 여부 | 내부망 + AES256 이상 |
| 운영성 (Operability) | 관리자 기능의 사용 편의성 | UI 기반 설정(템플릿, 역할, 워크플로우) 구성 여부 | 90% 이상 GUI 기반 |
|  | 백업/복구 기능 제공 | 요구 및 변경 데이터의 정기 백업 및 복구 기능 | 주간 자동 백업, 수동 복구 지원 |
| 접근성 (Accessibility) | 다양한 환경에서의 접근 가능 여부 | PC, 모바일, Web 브라우저 호환성 | 주요 브라우저 3종 이상 지원 |
|  | 접근성 품질 | 웹 접근성 및 직관적인 UI 설계 | 시각/이해 85점 이상 (내부 평가 기준) |
| 유지보수성 (Maintainability) | 기술지침 및 매뉴얼 제공 | 설치/운영/API 연계 가이드 제공 여부 | 100% 제공 |
|  | 버전 관리 및 배포 전략 | 업그레이드 시 중단 최소화 전략 | 무중단 배포 또는 사전 알림 기반 관리 |

## 제약조건

### 기술적 제약조건

|  |  |  |
| --- | --- | --- |
| 카테고리 | 설명 | 필수여부 |
| 네트워크 망 구성 | 내부망 전용 구성 또는 망 분리 운영 환경(업무망/인터넷망)이 요구됨. SaaS형 도구는 망연계 게이트웨이 필수 | ✔ 필수 |
| 서버 인프라 아키텍처 | On-Premise 구축 시 리눅스 기반 WAS + RDBMS 사용 필수. 클라우드 도입 시 내부 CSP 기준 준수 필요 (예: AWS, NCP 등) | ✔ 필수 |
| 보안/암호화 정책 | HTTPS 전송 필수, 저장 시 요구/사용자 정보 AES-256 이상 암호화 요구, SSO(SAML2.0 등) 연동 필요 | ✔ 필수 |
| API 연계 제한사항 | 외부 시스템 연동 시 REST API 기반만 허용. SOAP, Custom RPC 방식은 제한될 수 있음 | ✔ 필수 |
| 로그 보존 정책 | 변경/접속/승인 로그는 최소 1년 이상 보관되어야 하며, 조직 감사 정책 준수 필요 | ✔ 필수 |
| 데이터 위치 제한 | 모든 운영 및 백업 데이터는 국내 IDC 또는 내부 프라이빗 클라우드에 저장해야 함 (공공기관 필수조건) | ✔ 필수 |
| 브라우저 호환성 | IE는 지원하지 않으며, Chrome/Edge/Firefox 등 최신 브라우저 중심 지원이 필요함 | ❍ 선택 |
| 업그레이드 정책 | 내부 QA 완료 후 수동 업그레이드만 허용되며, 자동 업데이트 또는 강제 배포는 제한됨 | ❍ 선택 |
| 로그인 인증 구조 | 사내 통합인증(SSO), AD/LDAP 연동 필수. 외부 인증 서버 또는 클라우드 계정 사용은 제한될 수 있음 | ✔ 필수 |
| 서버 접근 통제 | 백엔드 서버 접근은 방화벽/IP 정책으로 통제되며, 벤더 접속 시 VPN 및 운영 승인 필요 | ✔ 필수 |

### 비즈니스 제약조건

|  |  |  |
| --- | --- | --- |
| 카테고리 | 설명 | 필수여부 |
| 예산 집행 방식 | 초기 투자 방식이 아닌 연간 사용료(SaaS 모델) 방식은 조직 예산집행 구조상 제한될 수 있음 | ✔ 필수 |
| 외부 클라우드 사용 제한 | 공공기관, 금융사 등은 민감 데이터의 외부 클라우드(해외 SaaS 포함) 사용 불허 정책 존재 | ✔ 필수 |
| 벤더 계약 조건 | 유지보수 계약 시 연간 1회 이상 기능 업그레이드 요구 또는 무중단 패치 조건은 제한될 수 있음 | ❍ 선택 |
| 공공기관 도입 기준 | 행정안전부의 공공 클라우드 보안 인증 또는 KISA CSAP 인증 도구만 사용 가능 | ✔ 필수 |
| 다부서 연동 구조 요구 | 단일 프로젝트 도구가 아닌 전사 운영을 위한 부서 간 공동 사용 필요 (관리 기준 통합 필요) | ✔ 필수 |
| 도입 일정 제약 | 회계연도 기준 예산 편성 마감 전 도입 완료 요구 (특히 3분기 이전) | ✔ 필수 |
| 사용자 교육 시간 확보 제한 | 사용자 대상 정기 교육 또는 실습 시간 확보가 어렵기 때문에 1주 이내 도입 정착 가능한 UI/UX 필요 | ❍ 선택 |
| RFP 고시 조건 포함 여부 | 특정 기능이 RFP 고시 조건에 포함되지 않으면 기능 도입이 불가능할 수 있음 (특히 조달청 기준) | ✔ 필수 |
| 조직 내 툴 표준화 정책 | 이미 Jira, Confluence 등 표준 도구가 존재하며, 신규 도입 시 이와 통합 또는 대체 설계 필수 | ✔ 필수 |
| 외부 감사 대응 조건 | 감사 대상 기준에 따라 로그 저장 기간, 승인 흐름, 변경사유 기록이 법적 대응 기준에 맞아야 함 | ✔ 필수 |

## 솔루션 평가

### 평가영역 및 가중치

|  |  |  |  |
| --- | --- | --- | --- |
| 평가영역 | 가중치 (%) | 주요 평가요소 | 세부 평가지표 |
| 기능적 적합성 | **35%** | 요구 수명주기 관리, 요구-테스트-릴리즈 트레이서빌리티, 승인 및 협업 기능 | 워크플로우 구성 유연성, 이력/변경 로그 제공, 역할 기반 승인 지원 여부 등 |
| 기술적 적합성 | **25%** | REST API 제공, DevOps 연계, 모델링 도구 연계, SSO/보안구조 | Jenkins/Git 연계, BPM/UML 연동 여부, 보안 인증 및 데이터 암호화 구조 등 |
| 운영 적합성 | **15%** | 사용자 UI/UX, 관리자 설정 편의성, 사용자 교육지원성 | GUI 기반 구성 기능, 실사용자 리뷰 기능, 사용성 평가점수(내부 기준 등) |
| 전략적 적합성 | **15%** | EA/프로세스 정합성, 감사 대응, 전사 적용 확장성 | ArchiMate/BPM 연계 가능성, 로그 감사 체계, 부서 간 사용 확장성 |
| 재정적 적합성 | **10%** | 도입 비용, 유지보수 비용, 장기 총소유비용(TCO) | 초기 라이선스 비용, 연간 유지비, 기술 지원 포함 여부 등 |

### 전략적 해석 가이드

|  |  |
| --- | --- |
| 조직 유형 | 평가영역 가중치 조정 예시 |
| **전사 통제 기반 조직 (EA 중심)** | 전략 정합성 ↑, 기능 완성도 ↑ |
| **DevOps·릴리즈 중심 조직** | 기술 연계성 ↑, 기능 완성도 ↑ |
| **품질·감사 대응 조직 (규제 산업)** | 기능 완성도 ↑, 전략 정합성 ↑ |
| **중견 프로젝트 중심 조직** | 운영 수용성 ↑, 기능 완성도 ↑ |
| **예산 제한 조직** | 재정 적합성 ↑, 운영 적합성 ↑ |

### 상세평가

#### 기능적 적합성 평가 (35%)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 평가요소 | 가중치 (%) | DOORS Next | Jira + Xray | Jama Connect | CodeBeamer | Polarion |
| 요구 수명주기 관리 (생성종료) | 10% | 9 | 7 | 8 | 9 | 9 |
| 승인 워크플로우 설정 및 역할기반 통제 | 10% | 10 | 7 | 8 | 9 | 9 |
| 요구사항 추적성 (Traceability Matrix) | 10% | 10 | 6 | 8 | 9 | 10 |
| 변경 이력 및 승인 로그 관리 | 5% | 10 | 6 | 8 | 9 | 10 |
| 협업 기능 (댓글, 알림, 리뷰 등) | 5% | 7 | 9 | 10 | 7 | 8 |
| 가중치 반영 점수 (35점 만점) | — | **9.45** | **6.9** | **8.65** | **8.9** | **9.45** |
| 도구 | **전략 분석** | | | | | |
| DOORS Next / Polarion | 기능 충실도 매우 높음. 요구 흐름, 승인, 변경, 추적, 협업 기능 균형 잡힘. 전사 통제형에 적합 | | | | | |
| CodeBeamer | 규제산업 대응 기능은 우수. 협업성과 UX는 보완 필요 | | | | | |
| Jama Connect | 실무 협업에 강점. 트레이서빌리티 완성도는 중상 | | | | | |
| Jira + Xray | 협업/운영성은 강하지만 통제기능, 승인 흐름, 추적성은 약함 | | | | | |

#### 기술적 적합성 평가 (25%)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 평가요소 | 가중치 (%) | DOORS Next | Jira + Xray | Jama Connect | CodeBeamer | Polarion |
| REST API, Webhook 제공 범위 | 7% | 9 | 10 | 9 | 9 | 9 |
| DevOps 연계 가능성 (Jenkins, Git 등) | 6% | 8 | 10 | 8 | 9 | 8 |
| 모델링 도구 연계 (SysML, BPMN, UML 등) | 5% | 9 | 5 | 6 | 8 | 10 |
| 보안 구조 대응 (SSO, 암호화, 접근통제 등) | 4% | 10 | 7 | 8 | 9 | 10 |
| 클라우드/온프렘 유연성 | 3% | 9 | 10 | 9 | 8 | 8 |
| 가중치 반영 점수 (25점 만점) | — | **8.85** | **8.75** | **8.15** | **8.8** | **9.1** |
| 도구 | **전략 분석** | | | | | |
| Polarion | EA 기반 모델링 연계 최적화, 보안 구조 대응성 우수. 전사 통합/보안 우선 조직에 적합 | | | | | |
| DOORS Next / CodeBeamer | 기술 연계 완성도 매우 높음. DevOps-모델링-보안 균형 잡힘 | | | | | |
| Jira + Xray | DevOps 친화도 최고 수준. 하지만 EA 기반 연계성, 보안 통제 등은 커스터마이징 필요 | | | | | |
| Jama Connect | 전반적 기술 연계 우수. 단, BPMN 등 모델 연계 기능은 보완 여지 있음 | | | | | |

#### 운영 적합성 평가(15%)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 평가요소 | 가중치 (%) | DOORS Next | Jira + Xray | Jama Connect | CodeBeamer | Polarion |
| 사용자 UI/UX 직관성 | 5% | 6 | 9 | 9 | 7 | 7 |
| 관리자 설정 편의성 (GUI 기반) | 4% | 7 | 9 | 8 | 7 | 7 |
| 사용자 커스터마이징 기능성 | 3% | 7 | 9 | 9 | 8 | 8 |
| 사용자 교육/도입 용이성 | 3% | 6 | 9 | 8 | 7 | 7 |
| 가중치 반영 점수 (15점 만점) | — | **6.55** | **9.0** | **8.7** | **7.15** | **7.2** |
| 도구 | **전략 분석** | | | | | |
| Jira + Xray | UX·설정·도입 속도에서 최고 수준. DevOps 및 실무 중심 조직에 매우 적합 | | | | | |
| Jama Connect | 협업 UI, 템플릿 구성 뛰어남. 관리자 설정도 직관적. 중견 조직에 최적화 | | | | | |
| Polarion / CodeBeamer | 통제 기능은 강하나 설정·UX에서 진입 장벽이 있음. 고도 통제형 조직에 적합 | | | | | |
| DOORS Next | 뛰어난 기능 대비 UX는 낮음. 관리자 중심 활용 가능하나 일반 사용자 도입은 어려움 | | | | | |

#### 전략적 적합성 평가 (15%)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 평가요소 | 가중치 (%) | DOORS Next | Jira + Xray | Jama Connect | CodeBeamer | Polarion |
| EA 아키텍처 정합성 (업무-시스템-데이터 구조 연계) | 6% | 9 | 6 | 7 | 8 | 10 |
| 전사 확산 가능성 (다부서 공동사용/템플릿화) | 4% | 8 | 9 | 9 | 8 | 9 |
| 전략 정책 일관성 대응 (감사/품질/보안 정책 반영) | 3% | 10 | 6 | 8 | 9 | 10 |
| 기술 확장성 및 버전 안정성 | 2% | 9 | 8 | 8 | 8 | 9 |
| 가중치 반영 점수 (15점 만점) | — | **9.05** | **6.95** | **8.2** | **8.3** | **9.7** |
| 도구 | **전략 분석** | | | | | |
| Polarion | EA 구조 정합성, 전략 정책 반영력 최고 수준. 전사 아키텍처 기반 체계화 가능 | | | | | |
| DOORS Next | 전략적 통제형 도입에 매우 강력. EA 기반 정책 연계 완성도 높음 | | | | | |
| CodeBeamer / Jama | 품질 및 정책 대응은 강력하나, 아키텍처 구조 연결성은 보완 가능성 존재 | | | | | |
| Jira + Xray | 실무 중심 전략에는 유리하나, 전사 EA 연계나 전략 일관성 측면은 약함 | | | | | |

#### 재정적 적합성 평가 (10%)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 평가요소 | 가중치 (%) | DOORS Next | Jira + Xray | Jama Connect | CodeBeamer | Polarion |
| 초기 도입 비용 (라이선스/구축 포함) | 3% | 6 | 9 | 8 | 7 | 6 |
| 연간 유지보수/서브스크립션 비용 | 3% | 7 | 9 | 7 | 8 | 7 |
| TCO (3년 기준 총소유비용) | 2% | 7 | 9 | 8 | 8 | 7 |
| 기술지원/교육 포함 여부 | 2% | 9 | 8 | 9 | 9 | 9 |
| 가중치 반영 점수 (10점 만점) | — | **7.0** | **8.9** | **8.1** | **8.2** | **7.6** |

### 전체 평가 요약

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 도구 | 기능(35) | 기술(25) | 운영(15) | 전략(15) | 재정(10) | 총점(100) |
| Polarion | 9.45 | 9.1 | 7.2 | 9.7 | 7.6 | **43.05 / 50.0 + 43.05% = 86.1** |
| DOORS Next | 9.45 | 8.85 | 6.55 | 9.05 | 7.0 | **40.9 + 40.9% = 81.8** |
| CodeBeamer | 8.9 | 8.8 | 7.15 | 8.3 | 8.2 | **41.35 + 41.35% = 82.7** |
| Jama Connect | 8.65 | 8.15 | 8.7 | 8.2 | 8.1 | **41.8 + 41.8% = 83.6** |
| Jira + Xray | 6.9 | 8.75 | 9.0 | 6.95 | 8.9 | **40.5 + 40.5% = 81.0** |