

TMMi® in the DevOps world (v1.1)

#SuccessWithTMMi

에디터: 에릭 반 비넨달(ERIK VAN VEENENDAAL)

TMMi는 특정 라이프사이클에 얽매이지 않도록 설계되었습니다. 본 문서는 TMMi 테스트 개선 모델이 DevOps 환경에서 어떻게 유용하게 활용되고 적용될 수 있는지를 설명합니다. TMMi 레벨 2와 레벨 3의 프로세스 영역 및 그 목표들이 하나씩 논의됩니다. 이들이 DevOps와 어떤 관련이 있는지, 그리고 관련 프랙티스들이 일반적으로 어떤 모습일지 기술합니다. 만약 특정 프랙티스가 DevOps 환경에서 수행될 필요가 없거나 부가적인 가치가 없다고 판단되는 경우, 이 역시 명시적으로 언급합니다. 본 문서는 완전한 DevOps 테스트 syllabus를 제공하려는 것이 아니라, TMMi 목표들이 DevOps 환경에서 어떻게 해석되어야 하는지를 '단지' 보여주고 향후 연구를 위한 아이디어와 방향을 제시하기 위함임을 유의해 주십시오. 또한, 'DevOps 환경에서의 TMMi'가 전체 TMMi 모델을 대체하는 설명서가 되도록 의도된 것도 아닙니다. 'DevOps 환경에서의 TMMi'는 항상 TMMi 참조 모델 문서와 함께 사용되어야 합니다.

© TMMi Foundation 2011–25

모든 권리 보유. 본 출판물의 어떠한 부분도 TMMi 재단의 사전 허가 없이는 관련 라이선스 문서에 명시된 방식을 제외하고는 어떤 형태나 수단으로든 전부 또는 일부를 임대, 판매, 양도, 복제하거나 전송할 수 없습니다. 관련 라이선스 문서의 조건에 따라 복사가 허용되는 경우에도, 이 고지 사항이 모든 복사본에 재생되어야 합니다.

상표로 간주될 만한 근거가 있는 용어들은 그렇게 지정되었습니다. 하지만 그러한 지정의 유무가 해당 상표의 법적 지위에 영향을 미치는 것으로 간주되어서는 안 됩니다.

TMMi는 TMMi Foundation (영국)의 등록 상표입니다.

기여자(Contributors)

Katalin Balla (Hungary)

Suresh Chandra Bose (USA)

Kari Kakkonen (Finland)

Mattijs Kemmink (The Netherlands)

Rik Marselis (The Netherlands)

Szilard Szell (Hungary)

Erik van Veenendaal (The Netherlands)

개정 이력

이 섹션은 본 문서가 배포될 때마다 주요 개정 사항을 요약하여 제공합니다.

이 내용은 정보 제공 목적으로만 제공됩니다.

배포	개정 노트
V1.0	"DevOps 환경에서의 테스트(Testing in the DevOps world)" 문서의 초기 버전입니다.
V1.1	DevOps 환경에서 TMMi 레벨 3 의 프로세스 영역, 목표 및 프랙티스에 대한 해석이 추가되었습니다. 또한, DevOps 관련 용어에 대한 용어집(Glossary)이 추가되었습니다.

Table of Contents

기여자(Contributors)	2
개정 이력.....	3
Table of Contents.....	4
1 TMMi® in the DevOps world	7
1.1 서론.....	7
1.2 DevOps and DevSecOps	7
1.3 테스트 성숙도 모델 통합 (TMMi)	8
1.4 DevOps 환경에서의 테스트	9
1.4.1 DevOps 환경에서 테스트의 과제	9
1.4.2 TMMi 와 DevOps	10
1.4.3 ISTQB DevOps syllabus 와 관계	12
2 TMMi Level 2 관리(Managed).....	12
2.1 프로세스 영역 2.1 테스트 정책 및 전략(Test Policy and Strategy)	12
2.1.1 SG1 테스트 정책 수립(Establish a Test Policy)	12
2.1.2 SG2 테스트 전략 수립(Establish a Test Strategy)	14
2.1.3 SG3 테스트 성과 지표 수립 (Establish Test Performance Indicators)	16
2.2 프로세스 영역 2.2 테스트 계획 (Test Planning)	17
2.2.1 SG1 위험 평가 수행 (Perform Risk Assessment)	18
2.2.2 SG1 테스트 접근 방식 수립 (Establish a Test Approach)	19
2.2.3 SG3 테스트 추정치 수립 (Establish Test Estimates)	20
2.2.4 SG4 테스트 계획 개발 (Develop a Test Plan)	22
2.2.5 SG5 테스트 계획에 대한 약속 확보 (Obtain commitment to the Test Plan)	22
2.3 프로세스 영역 2.3 테스트 모니터링 및 통제 (Test Monitoring and Control)	23
2.3.1 SG1 계획 대비 테스트 진행 상황 모니터링 (Monitor Test Progress against Plan)	24
2.3.2 SG2 계획 및 기대치 대비 제품 품질 모니터링 (Monitor Product Quality against Plan and Expectations)	25
2.3.3 SG3 시정 조치 종결 관리 (Manage Corrective Actions to Closure)	26
2.4 프로세스 영역 2.4 테스트 설계 및 실행 (Test Design and Execution)	27

2.4.1	SG1 테스트 설계 기법을 이용한 테스트 분석 및 설계 수행 (Perform Test Analysis and Design using Test Design Techniques).....	27
2.4.2	SG2 테스트 구현 수행 (Perform Test Implementation)	29
2.4.3	SG3 테스트 실행 수행 (Perform Test Execution).....	30
2.4.4	SG4 테스트 인시던트 종결 관리 (Manage Test Incidents to Closure)	31
2.5	프로세스 영역 2.5 테스트 환경 (Test Environment)	32
2.5.1	SG1 테스트 환경 요구사항 개발 (Develop Test Environment Requirements)	32
2.5.2	SG2 테스트 환경 구현 수행 (Perform Test Environment Implementation)	33
2.5.3	SG3 테스트 환경 관리 및 통제 (Manage and Control Test Environments)	33
3	TMMi Level 3 정의(Defined)	35
3.1	프로세스 영역 3.1 테스트 조직(Test Organization)	35
3.1.1	SG1 테스트 조직 수립(Establish a Test Organization)	35
3.1.2	SG2 테스트 전문가를 위한 기능 수립(Establish Test Functions for Test Specialists)	36
3.1.3	SG3 테스트 경력 경로 수립(Establish Test Career Paths)	36
3.1.4	SG4 테스트 프로세스 개선 결정, 계획 및 구현(Determine, Plan and Implement Test Process Improvements).....	37
3.1.5	SG5 조직 테스트 프로세스 배포 및 교훈 통합(Deploy the Organizational Test Process and Incorporate Lessons Learned)	38
3.2	프로세스 영역 3.2 테스트 교육 프로그램(Test Training Program)	39
3.2.1	SG1 조직 테스트 교육 역량 수립(Establish an Organizational Test Training Capability)	39
3.2.2	SG2 필요한 테스트 교육 제공(Provide Necessary Test Training)	39
3.3	프로세스 영역 3.3 테스트 라이프사이클 및 통합(Test Lifecycle and Integration)	40
3.3.1	SG1 조직 테스트 프로세스 자산 수립(Establish Organization Test Process Assets)	40
3.3.2	SG2 테스트 라이프사이클 모델을 개발 모델과 통합(Integrate the Test Lifecycle Models with the Development Models)	42
3.3.3	SG3 마스터 테스트 계획 수립(Establish a Master Test Plan)	42
3.4	프로세스 영역 3.4 비기능 테스트(Non-Functional Testing)	43
3.4.1	SG1 비기능 제품 위험 평가 수행(Perform a Non-Functional Product Risk Assessment)	43
3.4.2	SG2 비기능 테스트 접근 방식 수립(Establish a Non-Functional Test Approach)	44
3.4.3	SG3 비기능 테스트 분석 및 설계 수행(Perform Non-Functional Test Analysis and Design)	45
3.4.4	SG4 비기능 테스트 구현 수행(Perform Non-Functional Test Implementation)	46
3.4.5	SG4 비기능 테스트 실행 수행(Perform Non-Functional Test Execution)	46
3.5	프로세스 영역 3.5 동료 검토(Peer Reviews).....	47

3.5.1 SG1 동료 검토 접근 방식 수립(Establish a Peer Review Approach)	47
3.5.2 SG2 동료 검토 수행(Perform Peer Reviews)	49
DevOps 관련 용어집	51
References	53

1 TMMi® in the DevOps world

1.1 서론

이 문서는 소프트웨어 개발을 위한 DevOps 접근 방식을 TMMi 테스트 프로세스 개선 모델과 결합하는 것이 비즈니스 목표를 달성하기 위한 가능한 경로임을 설명합니다. 또한, TMMi가 DevOps 환경에서 어떻게 유익하게 사용되고 적용될 수 있는지, 그리고 TMMi 프랙티스를 구현하는 전통적인 테스트 접근 방식에 대한 검증된 DevOps 기반 대안을 제공합니다. 이 문서는 조직이 성장하고 프로젝트 압력이 증가함에 따라 종종 가시성을 잃는 핵심 테스트 프랙티스를 상기시켜 줌으로써, DevOps 여정의 성숙도와 관계없이 TMMi가 DevOps 조직을 어떻게 도울 수 있는지 다룹니다. 각 TMMi 프로세스 영역과 그 구체적인 목표들이 하나씩 논의됩니다. 이들이 DevOps와 어떻게 관련되는지, 그리고 관련 프랙티스가 일반적으로 어떤 모습일지 기술합니다. 만약 특정 프랙티스가 부가적인 가치가 없어 DevOps 환경에서 수행될 필요가 없다고 예상되는 경우, 이 역시 명시적으로 언급됩니다.

오늘날 지속적인 통합(CI), 지속적인 전달(CD), 그리고 자동화가 소프트웨어 개발을 이끄는 빠르게 변화하는 DevOps 환경에서는, 모든 단계에서 품질을 보장하는 것이 무엇보다 중요합니다. 테스트 성숙도 모델 통합(TMMi)은 테스트 프랙티스를 평가하고 개선하기 위한 구조화된 프레임워크를 제공합니다. DevOps 팀이 품질을 타협하지 않으면서 신속한 릴리스를 위해 노력함에 따라, TMMi를 통합하는 것은 테스트 성숙도를 높이고, 비즈니스 목표와 일치하며, 결과를 제공하기 위한 로드맵을 제공합니다. TMMi와 DevOps 간의 이러한 시너지는 높은 품질과 신뢰성 기준을 유지하면서 개발 속도를 높이는 균형 잡힌 접근 방식을 만듭니다.

이 문서는 다음과 같은 여러 대상 독자를 가집니다. 주요 그룹은 다음과 같습니다:

- 프로세스 성숙도를 유지하면서 DevOps로 전환하고자 하는 성숙한 전통적 조직
- 테스트 성숙도를 높이기 위해 TMMi 적용을 시작하고, 이와 병행하여 DevOps로 전환하고자 하는 덜 성숙한 조직
- 구조화된 애자일 테스트 프로세스를 가지고 있으며, 이제 DevOps로 전환하고자 하는 애자일 조직
- 성공적이고 성장하고 있는 DevOps 조직으로, 애자일의 이점을 유지하는 동시에 어느 정도의 프로세스 성숙도가 필요한 경우

이 문서는 원본 TMMi 모델과 독립적으로 사용되도록 의도되지 않았습니다. 이는 DevOps 환경에서 테스트 프로세스 개선을 수행하는 사람들에게 다양한 TMMi 목표와 프랙티스의 해석 및 의미에 대한 지침을 제공하기 위함입니다. 이 문서는 원본 TMMi 모델을 보완하며, 보충 문서로 사용되어야 합니다.

1.2 DevOps and DevSecOps

DevOps는 소프트웨어 개발팀과 IT 운영팀 간의 격차를 해소하는 것을 목표로 하는 철학이자 일련의 프랙티스입니다. 협업과 공동 책임 문화를 조성함으로써, DevOps는 소프트웨어의 개발, 배포 및 유지보수를 간소화하고자 합니다. 핵심 아이디어는 전통적인 개발팀과 운영팀 간의 사일로(silo)를 허물어, 소프트웨어를 더욱 효율적이고 효과적으로 제공하는 것입니다. 본질적으로 DevOps는 자동화와 지속적인 프로세스를 강조합니다. 자동화는 수동 작업을 최소화하여 오류를 줄이고 속도를 높이는 데 결정적인 역할을 합니다. 여기에는 지속적인 통합(CI) 및 지속적인 배포(CD) 프랙티스를 통해 코드 변경 사항 통합, 테스트 및 배포 프로세스를 자동화하는 것이 포함됩니다. 또 다른 중요한 강조점은 비즈니스 목표 및 고객과 긴밀하게

연계된 자율성, 리더십, 협업, 빈번한 실험, 짧은 개발 주기, 배포 빈도 증가, 그리고 더 신뢰할 수 있는 릴리스라는 DevOps 문화로의 변화가 필요하다는 것입니다.

DevOps 접근 방식은 더 빠른 시장 출시 시간, 팀 간의 개선된 협업, 더 높은 소프트웨어 품질 및 신뢰성, 그리고 확장성 및 유연성 증가를 포함하여 수많은 이점을 제공합니다. 또한 문제를 더 빨리 감지하고 해결할 수 있게 함으로써, DevOps는 다운타임을 줄이고 시스템 탄력성(resilience)을 향상시킵니다. 궁극적으로 DevOps는 기술 및 프로세스에 관한 것만큼이나 문화적 변화에 관한 것입니다. 이는 개발팀과 운영팀 모두가 공동의 목표를 향해 노력하고, 피드백을 수용하며, 지속적으로 개선을 위해 노력하는 사고방식의 전환을 필요로 합니다. 이러한 요소들을 통합함으로써, DevOps는 조직이 소프트웨어를 더욱 효율적이고 신뢰할 수 있으며 안전하게 제공하도록 돕습니다.

DevSecOps

DevSecOps는 Development(개발), Security(보안), Operations(운영)의 약자로, 보안 프랙티스를 DevOps 프로세스에 통합하는 접근 방식입니다. 이 방법론은 초기 개발부터 배포 및 그 이후까지, 전체 IT 라이프사이클 전반에 걸쳐 보안이 공동 책임이 되도록 보장하는 것을 목표로 합니다. DevSecOps는 보안 침해 위험을 줄이면서 더 빠르고 효율적으로 보안이 확보된 소프트웨어를 제공하는 것을 목표로 합니다.

1.3 테스트 성숙도 모델 통합 (TMMi)

TMMi 프레임워크는 테스트 관리자, 테스트 엔지니어, 개발자 및 소프트웨어 품질 전문가에게 중요한 문제들을 다루는 테스트 프로세스 개선을 위한 지침 및 참조 프레임워크로 TMMi 재단에 의해 개발되었습니다. TMMi 내에서 테스트는 모든 소프트웨어 제품 품질 관련 활동을 포괄하는 가장 광범위한 의미로 정의됩니다. TMMi는 프로세스 평가 및 개선을 위해 성숙도 레벨 개념을 사용합니다. 나아가, 프로세스 영역(process areas), 목표(goals), 그리고 프랙티스(practices)이 식별됩니다. TMMi 성숙도 기준을 적용하는 것은 테스트 프로세스를 개선하며, 제품 품질, 테스트 엔지니어링 생산성, 및 주기 시간(cycle-time) 노력에 긍정적인 영향을 미치는 것으로 나타났습니다. TMMi는 조직이 테스트 프로세스를 평가하고 개선하도록 지원하기 위해 개발되었습니다.

TMMi는 프로세스 개선을 위한 단계적 아키텍처(staged architecture)를 가지고 있습니다. 여기에는 조직의 테스트 프로세스가 임시적이고 관리되지 않는 상태에서 관리되고, 정의되며, 측정되고, 최적화되는 상태로 진화함에 따라 진행되는 단계 또는 레벨이 포함됩니다. 각 단계를 달성하는 것은 해당 단계의 모든 목표가 달성되었음을 보장하며, 그 개선 사항들은 다음 단계의 토대가 됩니다. TMMi의 내부 구조는 체계적인 방식으로 학습되고 적용될 수 있는 풍부한 테스트 프랙티스들을 포함하고 있으며, 이는 점진적인 단계로 개선되는 고품질 테스트 프로세스를 지원합니다. TMMi에는 성숙도 계층 및 테스트 프로세스 개선을 위한 진화 경로를 규정하는 다섯 가지 레벨이 있습니다. 각 레벨은 조직이 해당 레벨에서 성숙도를 달성하기 위해 구현해야 하는 일련의 프로세스 영역을 가지고 있습니다. TMMi의 각 성숙도 레벨에 대한 프로세스 영역은 그림 1에 나와 있습니다.

TMMi의 주요 기본 원칙은 다양한 라이프사이클 모델 및 환경에 적용 가능한 일반 모델(generic model)이라는 것입니다. TMMi에서 정의된 대부분의 목표 및 프랙티스는 순차적 라이프사이클 모델과 반복적 라이프사이클 모델(애자일을 포함) 모두에 적용 가능한 것으로 나타났습니다. 그러나 모델의 가장 낮은 수준에서는 제공되는 많은 하위 프랙티스 및 예시들이 적용되는 라이프사이클 모델에 따라 (매우) 다릅니다. TMMi 내에서는 오직 목표(goals)만 의무적이며, 프랙티스(practices)은 의무적이지 않다는 점에 유의해야 합니다. TMMi는 TMMi

재단 웹사이트에서 자유롭게 이용 가능하며 출판된 책 형태로도 제공됩니다. 이 모델은 스페인어, 프랑스어, 중국어를 포함한 여러 언어로 번역되었습니다.

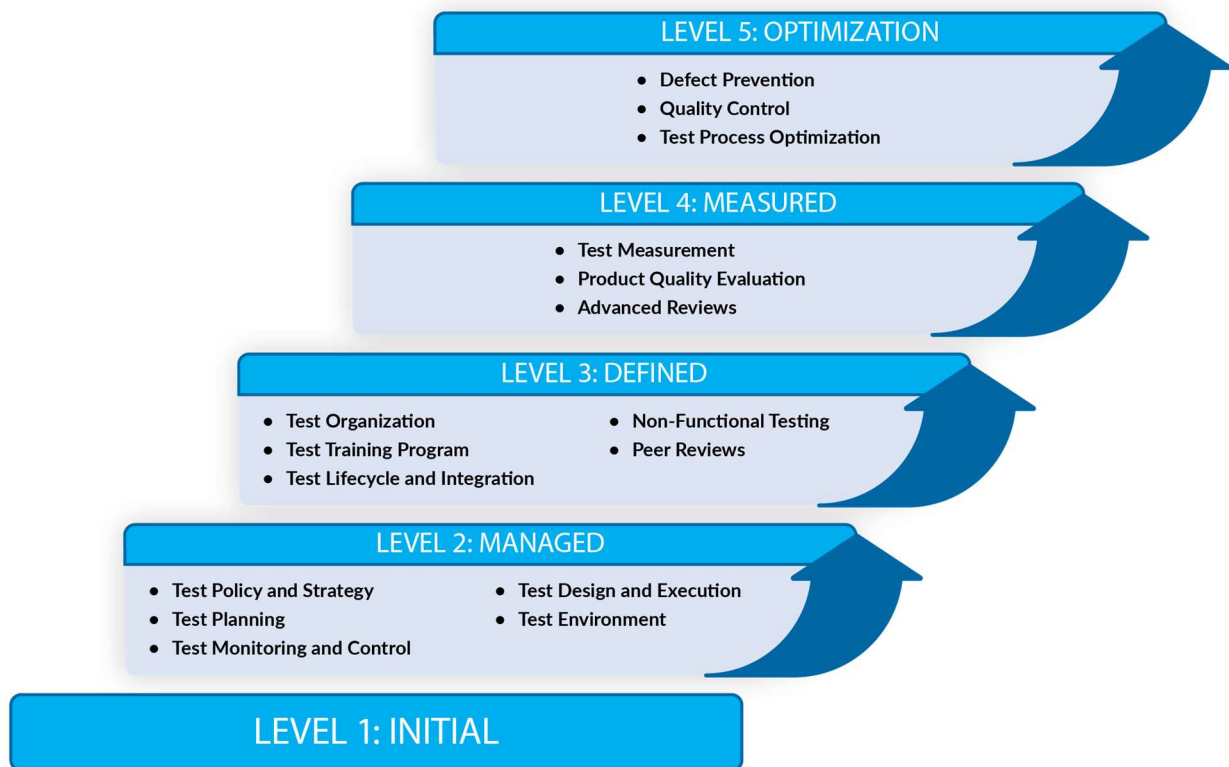


Figure 1: TMMi 성숙도 레벨 및 프로세스 영역

1.4 DevOps 환경에서의 테스트

1.4.1 DevOps 환경에서 테스트의 과제

DevOps 환경에서의 테스트는 지속적인 통합 및 지속적인 배포(CI/CD) 파이프라인의 특성, 개발 및 운영 팀 간의 협업, 그리고 빠른 개발 주기로 인해 고유한 과제를 제시합니다. 주요 테스트 과제 중 일부는 다음과 같습니다:

- **지속적인 테스트 (Continuous Testing):** CI/CD 파이프라인을 따라잡기 위해 테스트는 빠르고 자주 실행되어야 합니다. 견고하고 신뢰할 수 있는 테스트 스크립트와 프레임워크가 필요한 자동화된 테스트에 대한 의존도가 높습니다.
- **환경 관리 (Environment Management):** 지속적인 테스트를 위한 적절한 테스트 환경 관리 및 현실적인 테스트 환경 프로비저닝을 보장하는 것은 복잡할 수 있습니다. 종종 프로덕션 환경과 테스트 환경 간에 일치성(parity) 부족이 발생하여 미탐지된 문제로 이어질 수 있습니다. 또 다른 과제는 여러 환경에서 일관된

구성을 관리하고 유지하는 것입니다. 환경은 일반적으로 요청 시(on demand) 자동으로 배포 및 구성되어야 하며, 테스트 후에는 삭제되어야 합니다.

- *테스트 데이터 관리 (Test Data Management)*: 테스트 데이터는 프로덕션과 유사하고, 일관성이 있으며, 요청 시 배포되어야 하는 동시에, 개인 정보 보호 지침을 준수하는 것이 필수입니다.
- *통합 및 회귀 테스트 (Integration and Regression Testing)*: 복잡한 상호 의존성을 가진 여러 구성 요소와 서비스를 통합하여 테스트하는 것은 어렵습니다. 또한, 새로운 변경 사항이 기존 기능을 손상시키지 않도록 보장해야 하며, 이를 위해서는 포괄적인 회귀 테스트 스위트가 필요합니다.
- *도구 및 인프라 (Tooling and Infrastructure)*: 다양한 테스트 도구를 CI/CD 파이프라인 및 기타 DevOps 도구와 통합하고, 증가된 부하와 복잡성을 처리할 수 있도록 테스트 인프라가 확장 가능해야 합니다.
- *문화적 및 조직적 과제 (Cultural and Organizational Challenges)*: 전통적으로 사일로에서 운영되어 상충되는 인센티브를 가졌던 개발, 테스트 및 운영팀 간의 협업을 촉진하는 것입니다. 팀 구성원이 자동화된 테스트, 지속적인 통합 및 배포 프랙티스를 처리하는 데 필요한 기술을 갖추도록 보장해야 합니다.
- *성능 테스트 (Performance Testing)*: 통제된 테스트 환경에서 실제 부하 및 조건을 시뮬레이션하는 것 뿐만 아니라, 성능을 지속적으로 모니터링하고 최적화하기 위해 성능 테스트를 CI/CD 파이프라인에 통합해야 합니다.
- *보안 테스트 (Security Testing)*: 빠르게 움직이는 파이프라인으로 인해 보안 테스트가 종종 뒷전으로 밀려납니다. 개발 속도를 늦추지 않으면서 보안 테스트(DevSecOps)를 통합하고 취약점을 조기에 포착하는 것은 많은 팀에게 과제이며, 전문 도구와 전략이 필요합니다. 보안 테스트는 정적 및 동적 보안 테스트를 수행하는 자동화된 도구를 사용하여 개발 라이프사이클 초기와 전반에 걸쳐 통합되는 것이 바람직합니다.
- *피드백 및 보고 (Feedback and Reporting)*: 테스트 결과를 바탕으로 팀에게 빠르고 명확하며 실행 가능한 피드백을 제공해야 합니다. 빌드 품질에 대한 통찰력을 제공하는 포괄적인 보고서를 생성하면서도, 프로세스는 애자일하게 유지해야 합니다.
- *쉬프트 레프트 프랙티스 (Shift left practices)*: 라이프사이클에서 가능한 한 일찍 작업을 왼쪽으로 옮겨 품질을 개선합니다. 쉬프트 레프트 테스트는 (코드) 검토, 정적 분석, 철저한 단위 테스트 등의 수단을 통해 프로세스에서 더 일찍 테스트하는 것을 의미합니다.
- *쉬프트 라이트 프랙티스 (Shift Right practices)*: 프로덕션에 변경 사항을 가능한 한 자주 배포하는 것이 중요하므로, 테스트 전략은 CI(지속적인 통합) 부분의 테스트 활동과 CD(지속적인 배포) 부분의 프로덕션에서 시스템 사용 관찰 사이의 균형을 유지해야 합니다.
- *모니터링 및 피드백 루프 (Monitoring and Feedback Loops)*: DevOps 에서 테스트는 배포로 끝나지 않으며, 지속적인 모니터링과 프로덕션 환경으로부터의 피드백 수집이 중요합니다. 결함을 포착하고 성능을 최적화하기 위해 효과적인 모니터링 도구를 설정하고 프로덕션 데이터를 해석하는 것은 종종 간과되거나 미흡하게 개발됩니다.

1.4.2 TMMi 와 DevOps

TMMi 와 DevOps 접근 방식이 상충된다는 것은 잘못된 믿음입니다. DevOps 접근 방식과 TMMi 는 공존할 수 있을 뿐만 아니라, 성공적으로 통합될 경우 상당한 시너지 효과를 낼 수 있습니다. 또한, 테스트를 다르게 바라보는 것, 즉 테스트가 개발에 완전히 통합되고 CI/CD 파이프라인을 갖는 것이 '테스트' 개선 프로그램의 맥락에서 무엇을 의미하는지에 대한 과제도 있습니다. TMMi 의 'i'는 테스트가 소프트웨어 개발의 통합된 부분이어야 하며, 완전히 분리된 것으로 취급되어서는 안 된다는 사실을 의미합니다. TMMi 모델을 사용하면

제품 위험 분석과 같이 DevOps 환경에서 종종 '잊혀지는' 중요한 테스트 프랙티스를 상기시키는 역할을 합니다. 이 문서는 TMMi와 DevOps 방법론이 효과적으로 협력할 수 있음을 사례를 통해 보여줄 것입니다.

TMMi를 구현할 때, TMMi 모델의 의도는 조직에 일련의 프랙티스를 '강요'하는 것이 아니며, '준수 입증'을 해야 하는 표준으로 적용되어서도 안 된다는 점을 고려해야 합니다. 적절하게 사용될 경우, TMMi는 비즈니스 목표를 고려할 때 변화가 가치를 제공할 수 있는 특정 테스트 영역을 찾는 데 도움을 줄 수 있습니다. 이는 적용되는 라이프사이클 모델과 관계없이 사실입니다. TMMi 프랙티스는 기대되는 구성 요소이지만, 정의된 TMMi 프랙티스에 대한 '대안적' 프랙티스이라고 불리는 방식을 통해서도 달성될 수 있음을 항상 기억하는 것이 중요합니다. 항상 해당 프랙티스의 의도가 무엇인지, 그 근거는 무엇이며, 그것이 비즈니스에 어떻게 가치를 더하는지를 생각해야 합니다. DevOps 환경에서는 종종 의도가 대안적인 프랙티스를 통해 이미 달성되는 경우가 많습니다. 일반적으로, 비즈니스 요구에 의해 추진되는 한 '어떤' 해결책이든 적합합니다. TMMi를 사용할 때는 너무 규범적이지 않도록 해야 합니다. 이는 애초에 TMMi가 의도된 방식이 아닙니다. 항상 TMMi 목표와 프랙티스를 자신의 상황에 맞게 해석해야 합니다. 일반적으로, 특정 비즈니스 상황 내에서 프로세스 요구 사항을 먼저 설정함으로써, 프로세스 개선 우선순위를 집중하고 추진하는 방법에 대한 결정을 내릴 수 있습니다.

TMMi와 DevOps 사이에 발생하는 대부분의 충돌은, 구현 시 '좋은 프랙티스'이 어떠해야 하는지에 대한 역사적인 TMMi 관점이나, DevOps 프랙티스가 DevOps 방식 및 가치를 어떻게 지원해야 하는지에 대한 근거를 오해하는 데서 기인합니다. (선임 평가자를 포함한) TMMi 전문가들은 구현될 때 '좋은 TMMi 준수' 프랙티스가 어떤 모습이어야 하는지와 관련하여 의도치 않게 공유될 수 있는 메시지들을 재고하고 잠재적으로 다시 생각할 필요가 있을 것입니다. DevOps 접근 방식이 TMMi 프로세스와 적절하게 함께 구현될 때, 이는 테스트 프랙티스를 삭제하는 것이 아니라 효과적으로 구현하는 결과를 낳습니다. TMMi 내의 특정 (테스팅) 프랙티스 외에도 일반적인 프랙티스(generic practices)이 있다는 점에 유의해야 합니다. 일반적인 프랙티스의 목적은 프로세스 영역의 제도화(institutionalization)를 지원하는 것입니다. 이는 효과적으로 조직 내에 새로운 사람이 오거나 다른 변화가 발생할 때 해당 프로세스 영역을 지원할 인프라가 갖추어져 있도록 보장하는 것을 의미합니다.

전통적인 소프트웨어 개발 방식에서 DevOps로 전환하는 것은 현재 정의된 프로세스들을 가지치고 간소화(prune and lean)하려는 노력을 이끌어낼 수 있습니다. 이러한 방식으로 TMMi 기반 조직들은 DevOps에 모두 존재하는 애자일 및 린(lean) 사고방식으로부터 혜택을 얻을 수 있습니다. 사람들이 TMMi 모델에 없는 것들을 역지로 해석하여, 결과적으로 불필요하고 가치를 더하지 않는 프로세스 및 작업 산출물을 만들어내는 경향이 있었습니다. 모델의 근본과 개선 목표로 돌아가 TMMi 모델을 의도된 대로 사용함으로써, 조직은 실질적인 가치를 더하는 프로세스를 실제 프로세스 요구 및 목표와 일치시키는 것을 지원할 수 있습니다. 애자일/린 사고방식으로 프로세스를 가지치고 간소화하면, 사람들이 실제로 하는 일을 반영하는 프로세스가 만들어지고, 사용되는 데이터만 수집되도록 보장할 것입니다. 이러한 사고방식은 또한 가능한 한 일을 단순하게 유지하는 데 중점을 두게 되며, 이는 일반적으로 쉽지 않지만 TMMi 구현을 실천하는 사람들에게 이점을 가져다줄 것입니다. DORA 성능 매트릭스 (변경 리드 타임, 배포 빈도, 변경 실패율, 배포 실패 복구 시간)에 의해 안내되는[DORA16],[DORA24] DevOps 내의 개선은 일반적으로 신속한 조치를 취할 수 있는 작고 권한이 부여된 팀을 통해 이루어지며, 이는 TMMi가 DevOps 적용으로부터 혜택을 얻을 수 있는 또 다른 방법입니다. TMMi가 포괄적이기는 하지만, 조직이 TMMi의 모든 요소를 전부 적용할 수는 없습니다. 성공을 위해서는 조직이 자신들의 핵심 테스트 프랙티스와 우선순위 및 집중이 필요한 개선 사항을 식별해야 합니다.

1.4.3 ISTQB DevOps syllabus 와 관계

TMMi 와 ISTQB 는 소프트웨어 테스트 전문가 직업을 함께 더욱 홍보하기 위한 제휴를 맺었습니다. 또한, 이 문서가 개발되는 동안 TMMi 기술 위원회와 ISTQB “Quality in DevOps” syllabus 작업팀은 협력했습니다. TMMi 는 테스트 개선 모델로서, TMMi 모델의 구조를 기반으로 하는 미리 정의된 개선 접근 방식과 우선순위를 제공합니다. 본 문서의 초점은 DevOps 환경에서 일하는 사람들을 위해 TMMi 개선 목표에 대한 해석을 제공하는 것입니다. 이는 우수한 DevOps 엔지니어링 프랙티스에 대한 자세하고 완전한 설명을 제공하지는 않습니다. ISTQB “Quality in DevOps” syllabus 은 내용 기반 문서로서, 품질 엔지니어링 (테스팅 포함) 프랙티스와 같은 우수한 DevOps 엔지니어링 프랙티스에 대한 자세한 설명을 제공하는 것을 목표로 합니다. 따라서 이 두 문서는 서로 매우 보완적입니다. 본질적으로: TMMi 는 무엇을 개선해야 하는지 (어떤 프로세스를)를 명시하고, ISTQB DevOps syllabus 는 어떻게 해야 하는지 (엔지니어링 모범 사례를)를 설명합니다.

2 TMMi Level 2 관리(Managed)

2.1 프로세스 영역 2.1 테스트 정책 및 전략(Test Policy and Strategy)

테스트 정책 및 전략 프로세스 영역의 목적은 테스트 활동(예: 테스트 유형 및 테스트 레벨)이 명확하게 정의된 테스트 정책과 조직 전체 또는 프로그램 전체의 테스트 전략을 개발하고 수립하는 것입니다. 테스트 성과를 측정하고, 테스트 활동의 가치를 파악하며, 개선 영역을 드러내기 위해 테스트 성과 지표가 도입됩니다.

2.1.1 SG1 테스트 정책 수립(Establish a Test Policy)

테스트 개선 프로젝트를 시작하는 모든 조직은 테스트 정책을 정의하는 것으로 시작해야 합니다. 테스트 정책은 조직의 전반적인 테스트 목표, 목적, 그리고 테스트 및 테스트 전문가에 대한 전략적 관점을 정의합니다. 테스트 정책은 조직의 전체 비즈니스 및 품질 정책과 일치하는 것이 중요합니다. 테스트 개선은 명확한 비즈니스 목표에 의해 추진되어야 하며, 이는 테스트 (개선) 정책에 문서화되어야 합니다. 테스트 정책은 조직 내 모든 이해관계자들 사이에서 테스트 및 그 목적에 대한 공통된 관점을 얻는 데 필요합니다. 이 공통된 관점은 조직 전체의 테스트 (프로세스 개선) 활동을 정렬하는 데 필수적입니다. 테스트 목표는 그 자체로 목적이 되어서는 안 되며, 작동하는 소프트웨어와 제품 품질을 확립하려는 더 높은 수준의 목표에서 파생된다는 점에 유의해야 합니다.

위에 언급된 내용은 DevOps 문화를 수용하고 DevOps 소프트웨어 개발을 실행하는 조직에서도 마찬가지로 사실입니다. 실제로 많은 조직에서는 테스트 역할의 변화, 테스트의 독립성, 테스트 자동화, 그리고 DevOps 소프트웨어 개발에서 전문 테스터의 역할 등에 대한 많은 논의가 이루어지고 있습니다. 이러한 항목들은 경영진 및 기타 이해관계자들 간의 논의에서 다루어져야 하며 테스트 정책에 문서화되어야 할 전형적인 이슈들입니다. DevOps 를 실천하는 조직을 포함하여, 테스트 개선 프로젝트를 시작하려는 모든 조직은 그러한 이니셔티브를 위한 비즈니스 동인과 요구 사항을 식별하고 정의해야 합니다. 그렇지 않으면 개선 프로젝트를 시작할 이유가 무엇이겠습니까? 진정한 비즈니스 요구 사항을 파악하는 데 시간을 투자함으로써, 조직은 (테스트) 개선

우선순위를 어디에 집중할지, 예를 들어 어떤 프로세스 영역에 집중할지에 대한 결정을 내릴 수 있는 맥락을 제공받게 됩니다. 테스트 정책은 일반적으로 조직 차원의 문서, 웹 페이지 또는 벽 차트 형태이며, 프로젝트 차원의 문서가 아니라는 점에 유의하십시오.

TMMi의 구체적인 목표인 테스트 정책 수립은 그 구체적인 프랙티스를 포함하여, DevOps 소프트웨어 개발을 적용하는 조직에도 완전히 적용 가능합니다. 하지만, DevOps에서는 개발, 테스트, 운영이 일반적으로 동일한 프로세스 내에서 다루어지거나 적어도 긴밀하게 정렬됩니다. 따라서 테스트 정책은 단순한 테스트보다는 품질 엔지니어링에 더 초점을 맞춘 전반적인 DevOps 정책의 필수적인 부분이 될 것으로 예상됩니다. DevOps에서의 테스트는 Shift Left(선행) 및 Shift Right(후행) 접근 방식을 균형 있게 활용하고, 협업적 프랙티스와 Test First 사고방식을 사용하여 조직됩니다. DevOps에서 테스트 활동은 (DevOps 핸드북에 정의된) 세 가지 방식, 즉 흐름(flow), 피드백(feedback), 지속적인 실험과 학습을 모두 지원해야 합니다.

언급된 바와 같이, 테스트 목표는 Shift Left와 Shift Right 사이의 명확한 균형에 초점을 맞추어야 합니다.

Shift Left: 필요한 품질을 구축하기 위한 사전 예방적, 설계 및 테스트 우선(test first) 사고방식과, 배포 전(CI/CD) 파이프라인 내부 및 주변)에 결함을 포착하기 위한 테스트 활동을 포함합니다.

Shift Right: 결함이 침투하는 경우 프로덕션에서 서비스 다운타임을 줄이기 위한 반응적 조치를 포함하며, 예를 들어 다운타임을 줄이는 자동 롤백 메커니즘, 영향을 줄이는 블루-그린 배포(blue-green deployment), 또는 카나리 릴리스(canary release) 등이 있습니다.

테스트 활동의 목표는 결함(결함이 있는 소프트웨어)을 프로덕션에 배포할 확률을 줄이는 것이지만, 배포 전략은 주어진 실패의 영향을 줄이기 위해 적용되며, 이는 전반적인 품질 향상을 가져옵니다.

영향 감소를 달성하기 위해, 배포 파이프라인의 테스트 활동은 전반적인 위험 수준을 낮게 유지하면서도, 흐름(flow)과 리드 타임을 개선하도록 간소화되어야 합니다.

소프트웨어 개발 전반의 테스트 외에도, Shift Right (프로덕션에서의 테스트) 또한 DevOps에서 일반적인 프랙티스며, 이는 정책에 정의된 테스트 목표의 정의에도 영향을 미칩니다. Shift Right는 텔레메트리 정보를 기반으로 프로덕션에서 발생하는 이슈에 신속하게 대응하는 데 중점을 두어 다운타임을 줄이고, 가용성을 향상시켜, 결과적으로 품질을 개선합니다. 사이트 신뢰성 엔지니어링 (SRE) 프랙티스는 시스템, 서비스 및 애플리케이션의 상태를 모니터링하고, 상태 문제/편차에 대응하기 위한 폐쇄 루프 자동화 메커니즘을 생성하는 것을 포함합니다. 이는 또한 self-healing 라고도 불립니다.

DevOps에서는 예상되는 변경 사항이 많기 때문에, 테스트 목표는 기능적 측면뿐만 아니라 비기능적 측면에서도 회귀 테스트에 크게 중점을 둡니다. 일부 비기능적 측면은 배포 후 프로덕션에서 측정될 수 있지만 (Shift Right), 보안과 같은 다른 측면들은 프로덕션으로 넘어가기 전에 테스트될 필요가 있습니다.

테스트 목표 예시:

- 테스트는 변경 사항이 예상되는 가치를 제공하는지 (인수 테스트) 확인하는 것과, 기존 기능을 손상시키지 않는지 (회귀 테스트) 확인하는 것에 중점을 둡니다.
- 테스트는 변경 실패율 감소 또는 배포 빈도 증가와 같은 조직의 개발 목표를 지원해야 합니다.

또한, 테스트 정책에 따른 독립성과 상위 수준의 프로세스 정의는 일반적으로 DevOps 환경에서 영향을 받습니다. 독립성과 관련하여, 테스트는 개발 조직 내의 한 역할이며, 이중 역할을 나타내기 위해 종종 DevTest 라고 불립니다. 따라서 독립성은 일반적으로 낮습니다. 조직은 보통 SDET(Software Development Engineers in Testing)를 포함하는 소위 플랫폼 팀에 의존하는데, 이들의 주된 역할은 개발과 테스트의 겹적을

나타내기 위해 역할이 사용할 수 있도록 제품의 테스트 용이성 측면(testability aspects)과 테스트 자동화 솔루션을 개발하고 유지보수하는 것입니다. 또한, 비기능 테스트이나 복잡한 종단 간(end-to-end) 시나리오와 같은 더 복잡한 영역을 위해 일부 전담 테스트 팀이 지정될 수도 있습니다. 이 경우 독립성은 더 높지만, 여전히 협력은 긴밀합니다. 상위 수준 테스트 프로세스 정의와 관련하여, 테스트는 이제 개발 프로세스 또는 CI/CD 전략의 필수적인 부분이 되었으므로, 테스트 프로세스 역시 그러할 것입니다.

일반적으로 간결한/린(lean) 문서 형태인 테스트 정책은 조직 전체/가치 흐름(value stream)/애자일 릴리스 트레인(Agile release train)/트라이브(tribe)에 배포되도록 보장하는 것이 중요합니다. 테스트는 애자일 및 DevOps 팀 업무의 핵심적인 부분이며 별도의 조직이 아니므로, 정책은 관련되고 영향을 받는 모든 사람에게 알려져야 합니다.

2.1.2 SG2 테스트 전략 수립(Establish a Test Strategy)

테스트 전략은 테스트 정책을 따르며 프로젝트 내 테스트 활동의 시작점 역할을 합니다. 테스트 전략은 일반적으로 조직 전체 또는 프로그램 전체 수준에서 정의됩니다. 전형적인 테스트 전략은 상위 수준의 제품 위험 평가를 기반으로 하며, 수행될 테스트 유형 및 테스트 레벨에 대한 설명을 포함합니다. 단순히 각 프로젝트 내에서 통합 테스트와 인수 테스트가 수행될 것이라고 명시하는 것만으로는 충분하지 않습니다. 우리는 단위 테스트와 인수 테스트가 무엇을 의미하는지, 이것들이 일반적으로 어떻게 수행되며 주요 목표가 무엇인지를 정의할 필요가 있습니다. 경험에 따르면, 테스트 전략이 정의되고 준수될 때, 다양한 테스트 활동 간의 중복이 발생할 가능성이 적어져 보다 효율적인 테스트 프로세스로 이어집니다. 또한, 다양한 테스트 유형과 레벨의 테스트 목표 및 접근 방식이 정렬되므로, 누락될 가능성이 적어지고, 이는 보다 효과적인 테스트 프로세스와 결과적으로 더 높은 수준의 제품 품질로 이어집니다.

테스트 전략은 DevOps 환경 내에서 매우 중요한 문서입니다. 이는 주로 CI/CD 전략 내에서 팀이 수행해야 할 테스트를 상위 수준에서 정의하며, 어떤 테스트 레벨과 테스트 유형이 실행되고 그 접근 방식이 무엇인지 상위 수준에서 정의합니다. 각 테스트 레벨 및 테스트 유형에 대해 목표, 책임, 주요 지속적인 테스트 작업, 수락 기준/품질 게이트가 정의됩니다. 테스트 전략은 일반적으로 애자일 테스트 사분면(Agile Testing Quadrants)을 따르지만 (그림 2 참조), 파이프라인 내 활동의 흐름을 나타내기 위해 CI/CD 파이프라인을 중심으로 구축됩니다.

파이프라인은 CI/CD 도구에 의해 실행되는 활동뿐만 아니라, 팀이 백로그 항목에 대해 작업할 때 수행하는 모든 활동으로 이해되어야 합니다. 검토 및 탐색적 테스트와 같은 테스트 활동은 CI/CD 도구 외부에서 발생하지만, 이는 흐름/파이프라인/가치 흐름의 필수적인 부분으로서 발생합니다. 파이프라인은 기술/도구뿐만 아니라, 활동, 프로세스 및 사람들의 시각적 표현으로도 사용될 수 있습니다 (그림 3 참조).

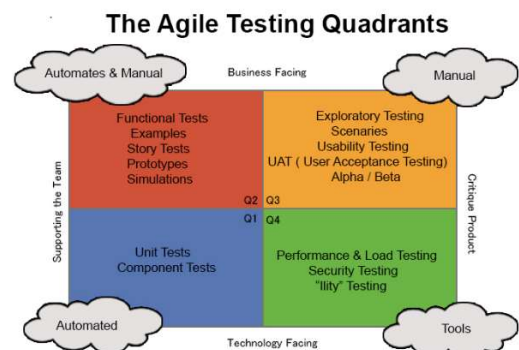


Figure 2: The Agile Testing Quadrants

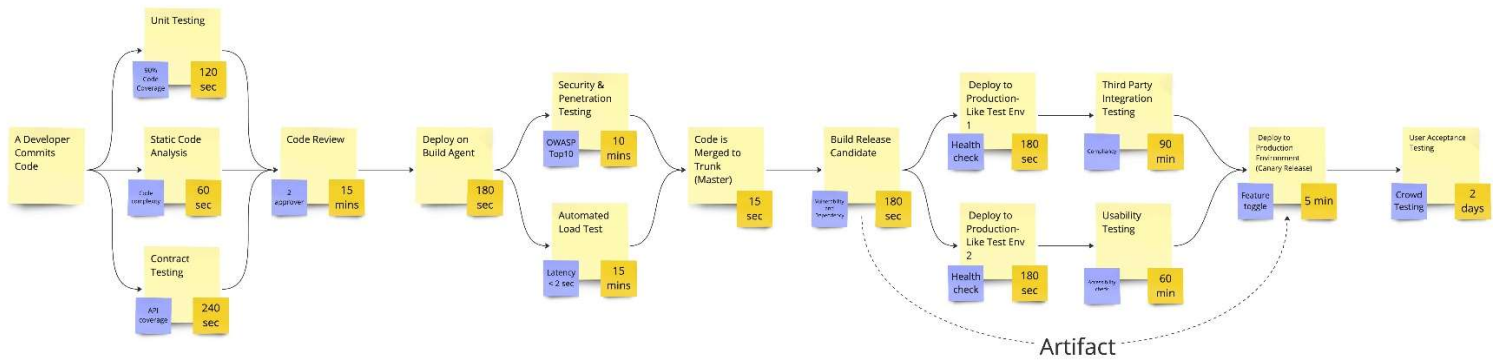


그림 3: 상위 수준 파이프라인으로 표현된 테스트 전략

DevOps 방식으로 작업할 때도 테스트 전략은 필수적인 문서입니다. DevOps 가 많은 조직에게 비교적 새롭기 때문에, 공통의 조직 전체 테스트 전략은 관련된 모든 사람을 안내하고 파이프라인 내에서 테스트가 어디서 어떻게 수행되는지 설명할 것입니다. 또한 각 프로젝트/파이프라인이 수레바퀴를 재발명하는 것(불필요한 중복 작업)을 방지할 것입니다.

테스팅은 개발 파이프라인의 통합된 부분이므로, 테스트 전략은 당연히 개발 전략과 완전히 정렬되어야 합니다. DevOps 테스트 전략을 개발할 때 고려해야 할 몇 가지 사항(주제별로 정리)은 다음과 같습니다:

테스트 레벨 (Test levels)

- 테스트 레벨은 주로 CI/CD 파이프라인 내의 테스트 활동/작업으로 표현됩니다.
- 테스트 전략은 종종 행동 주도 개발(BDD) 프랙티스에 의해 주도되는, 개발된 사용자 스토리의 인수 테스트를 중심으로 구축됩니다. 이는 파이프라인의 각 단계에서 지속적인 테스트를 요구합니다.
- 테스트 전략은 오픈 API 사용 위험을 완화하고 통합 테스트를 지원하기 위해 계약 테스트와 같은 테스트 레벨을 정의하는 것이 일반적입니다.

하위 수준 테스트 (Low level testing)

- 풀 리퀘스트(pull request), 병합(merging), 코드 리뷰 프랙티스와 같은 하위 수준 개발 및 테스트 프랙티스를 고려해야 합니다.
- 정적 테스트 및 기타 하위 수준 테스트 활동은 일반적으로 CI/CD 도구에 의해 의무화되며 종종 피할 수 없습니다 (예: 풀 리퀘스트의 일부로 자동 확인 및 테스트).
- 쉬프트 레프트 패러다임을 기반으로 단위 테스트, 합의된 코드 커버리지 수준 달성, 그리고 통합 테스트에도 중점을 둡니다.

테스트 자동화 (Test Automation)

- 지속적인 테스트는 소프트웨어 전달 파이프라인과 DevOps 도구 체인에 원활하게 통합됩니다.
- 테스트 자동화 전략은 DevOps 테스트 전략의 중요한 부분입니다.

진입 및 종료 기준 (Entry and Exit Criteria)

- 진입 기준(Entry Criteria)은 DevOps 팀을 위한 준비 완료 정의(Definition-of-Ready, DoR)로 표현됩니다. DoR 은 테스트의 입력으로 사용되는 고품질 테스트 기반을 의무화해야 합니다. 예를

들어, 팀이 협력하여 정의하고 일반적으로 Given-When-Then 문 형식으로 정의된 인수 기준(Acceptance Criteria)이 있습니다. 준비 상태에 도달하기 전에 위험과 테스트 목표가 잘 정의되고 합의되어야 하므로, 추정치에 테스트 요구 사항도 포함되도록 하는 것이 중요합니다.

- 종료 기준(Exit Criteria)은 DevOps 팀을 위한 완료 정의(Definition-of-Done, DoD)와 CI/CD 파이프라인의 모든 테스트 작업에 내장된 품질 게이트로 나뉩니다. DoD 는 테스트 충족도, 도달한 신뢰 수준 등을 나타내기 위해 다양한 백로그 레벨(에픽, 기능, 스토리)에 따라 다르게 정의될 수 있습니다. 성공적인 인수 테스트 외에도, DoD 는 일반적으로 모든 백로그 항목에 대한 제약 조건 역할을 하는 비기능 요구 사항을 충족하기 위한 예상되는 테스트 활동을 포함합니다. DevOps 팀에서 '완료(DONE)' 상태는 백로그 항목이 이미 프로덕션에 배포되어 시스템/서비스 모니터링을 기반으로 작동할 때만 달성됩니다.

테스트 유형: 보안 테스트 (Test Types: Security Testing)

- 성공에 중요한 비기능적 특성(예: 보안, 성능, 신뢰성 또는 유용성)과 그 테스트 접근 방식을 식별합니다.
- 보안의 중요성을 고려하여, 정적 애플리케이션 보안 테스트(SAST), 동적 보안 분석 도구(DAST), 대화형 애플리케이션 보안 테스트(IAST), 소프트웨어 구성 분석(SCA), 침투 테스트 및 API 보안 테스트가 보안 테스트 접근 방식의 (일부로) 언급될 수 있습니다.

테스트 환경 (Test Environments)

- 테스트 환경은 CI/CD 파이프라인의 다양한 테스트 단계(예: 개발, 스테이징, 사전 프로덕션)에 대해 표시되며, 각 단계는 다른 초점을 나타냅니다. 이들은 일반적으로 식별된 다양한 테스트 레벨(예: 단위, 통합, 시스템 및 인수 테스트)과도 정렬됩니다.
- DevOps 가 구성 코드화(Configuration-as-Code)와 같은 프랙티스를 적용하여 프로덕션 환경의 배포 및 구성 자동화에 중점을 두기 때문에, 테스트 환경을 설계하고 구축할 때 동일한 프랙티스를 사용하는 것이 당연합니다.
- 클라우드 기반 환경을 사용하면, 요청 시 테스트 환경을 생성하고 사용 후 삭제하는 것이 일반적인 프랙티스며, 이는 표준화되고 통제되며 안정적인 테스트 환경을 만듭니다.
- 블루-그린 배포와 같은 배포 전략을 활용할 때, 제품의 최신 버전 테스트 환경은 품질 게이트 및 DoD 를 성공적으로 충족한 후 프로덕션 환경이 될 수 있습니다. 이 경우 이전 프로덕션 환경은 다음 테스트 환경으로 재사용됩니다 (블루 환경과 그린 환경 간 목적 전환).

간결한 테스트 전략 문서는 종종 DevOps 조직에게 좋은 해결책이며, 상세한 (프로젝트) 테스트 계획에서 벗어날 수 있는 방법입니다. TMMi 의 구체적인 목표인 테스트 전략 수립(Establish a Test Strategy)은 그 구체적인 프랙티스를 포함하여, DevOps 소프트웨어 개발을 적용하는 조직에도 완전히 적용 가능합니다. 테스트 정책과 마찬가지로, 테스트 전략이 조직 전체 / 가치 흐름 / 애자일 릴리스 트레인 / 트라이브에 배포되도록 보장하는 것이 중요합니다.

2.1.3 SG3 테스트 성과 지표 수립 (Establish Test Performance Indicators)

테스트 정책에서 정의된 테스트 개선을 위한 비즈니스 목표는 일련의 핵심 테스트 성과 지표로 변환되어야 합니다. 테스트 정책과 이에 수반되는 성과 지표는 명확한 방향을 제시하며, 예상되는 테스트 성과 수준과 달성된 테스트 성과 수준을 소통하는 수단을 제공합니다. 이러한 성과 지표는 이해관계자들에게 테스트 및 테스트 프로세스 개선의 가치를 보여주어야 합니다. 프로세스 개선에 대한 투자는 장기적인 경영진의 지원이 필요하므로, 이들의 동기 부여를 유지하기 위해 개선 프로그램의 이점을 정량적으로 측정하는 것이 중요합니다. 이 TMMi의 구체적인 목표는 제한된 수(예: 2~3 개)의 테스트 성과 지표를 정의하는 것임에 유의하십시오. 이는 완전한 측정 프로그램을 설정하고 구현하는 것이 아니라, 테스트의 가치가 시간 경과 및 다양한 제공 환경에서 어떻게 변화하는지를 알려주는 핵심 지표 세트를 정의하는 것에 관한 것입니다.

애자일과 마찬가지로, DevOps 에서도 초점은 팀 기반 및 시스템 사고(systems thinking)에 더 맞춰질 것입니다. 이로 인해 지표는 테스트 자체의 세부 사항에만 국한되지 않고, 개발 및 배포 프로세스 성과 지표로 확장되어 폭넓게 사용될 수 있습니다.

주요 DevOps 성과 지표는 일반적으로 DORA 메트릭스(DevOps Research and Assessment Metrics)를 따릅니다. (테스트) 커버리지 대신, 배포 시 변경 실패율이 측정됩니다 (실패한 배포의 백분율).

DORA 메트릭스 예시 (Examples of DORA metrics):

- 소프트웨어 제공 성과 메트릭스 (Software delivery performance metrics): 배포 빈도(deployment frequency), 변경 리드 타임(lead time for changes), 서비스 복원 시간(time to restore service), 그리고 변경 실패율(change failure rate).
- 운영 성과 (Operational Performance): 신뢰성 메트릭스, 예: MTTR(평균 복구 시간) 및 MTTF(평균 고장 시간).

성과 지표는 CI/CD 활동의 필수적인 부분이어야 하며, 데이터는 거의 실시간으로 그리고 조직 내 모든 사람에게 제공되어야 합니다. DORA 메트릭스 외에도, 가치 흐름 관리/최적화를 지원하기 위해 흐름 메트릭스(Flow metrics)도 사용됩니다. 테스트 활동은 위 메트릭스에 맞춰 측정될 수 있으며, 그 기여도가 이해될 수 있도록 보장해야 합니다.

테스팅과 관련된 가치 흐름 관리를 위한 흐름 메트릭스 예시:

- 테스트 실행 리드 타임, 코드 변경당 CI/CD 내에서의 테스트 활동 성공률 또는 실패율.
- 흐름 메트릭스, 즉 테스트 실행 리드 타임이 전체 리드 타임에 어떻게 기여하는지.

과제는 TMMi 기반 테스트 개선 프로그램의 성과 달성을 잘 나타내는 동시에, DevOps 접근 방식 및 시스템 사고와 관련된 지표들을 적절히 혼합하여 정의하는 것일 것입니다. TMMi의 구체적인 목표인 테스트 성과 지표 수립은 그 구체적인 프랙티스를 포함하여 완전히 적용 가능하지만, 선택 및 적용되는 성과 지표는 테스트에만 관련된 것보다 더 넓은 범위와 폭을 가질 수 있습니다. 물론 후자는 성과 지표의 분석 및 해석을 더 어렵게 만들 것입니다. 실제로 사용되는 성과 지표가 '테스트 성과 지표'라고 불리지 않을 수도 있습니다. 그러나 테스트 관련 요소를 포함하고 있으며 테스트 개선 진행 상황을 평가하는 데 사용되는 한, TMMi의 맥락에서는 여전히 괜찮습니다.

2.2 프로세스 영역 2.2 테스트 계획 (Test Planning)

테스트 계획의 목적은 식별된 위험과 정의된 테스트 전략을 기반으로 테스트 접근 방식을 정의하고, 테스트 활동을 수행하고 관리하기 위한 충분한 근거가 있는 계획을 수립하고 유지하는 것입니다.

성공적인 테스트 계획의 핵심은 연관된 테스트 계획서("문서")를 정의하는 것이 아니라 사전 사고("활동")에 있다는 점에 유의하십시오.

DevOps 라이프사이클의 경우, 일반적으로 릴리스 계획과 반복(iteration) 계획의 두 가지 종류의 계획이 발생합니다. TMMi 레벨 2의 테스트 계획 프로세스 영역은 릴리스 계획과 반복 계획 모두에서의 테스트 관련 활동에 중점을 둡니다. 릴리스 계획은 프로젝트 시작 시점에 제품의 릴리스를 내다봅니다. 릴리스 계획에는 정의된 제품 백로그가 필요하며, 더 큰 사용자 스토리를 더 작은 스토리들로 다듬는 작업이 포함될 수 있습니다. 그러나 DevOps와 애자일은 일반적으로 고정된 시간과 유연한 내용으로 작업한다는 점을 명심해야 합니다. 테스트 접근 방식과 완료 정의(DoD)에 따라 테스트를 거친 백로그 항목만이 릴리스되도록 보장하는 것이 중요합니다. 릴리스 계획은 모든 반복을 아우르는 테스트 접근 방식과 테스트 계획의 기초를 제공합니다. 릴리스 계획은 상세한 작업보다는 전반적인 전략 또는 접근 방식에 초점을 맞춘 상위 수준의 계획입니다. 릴리스 계획이 완료된 후에는 첫 번째 반복을 위한 반복 계획이 시작됩니다. 반복 계획은 단일 반복의 끝을 내다보며 반복 백로그와 관련이 있습니다.

2.2.1 SG1 위험 평가 수행 (Perform Risk Assessment)

전체적인 테스트(Exhaustive testing)는 불가능하며, 선택은 항상 이루어져야 하고 우선순위는 항상 설정되어야 합니다. 따라서 이 TMMi 목표는 DevOps 프로젝트에도 적용 가능합니다. 반복 계획 동안 DevOps 팀은 다가오는 반복에서 구현될 사용자 스토리를 기반으로 제품 위험을 식별하고 분석합니다. 가급적 애자일 팀의 모든 구성원과 가능하면 일부 다른 이해관계자들이 제품 위험 세션에 참여합니다. 그 결과는 테스트를 위한 중요한 영역을 식별하는 우선순위가 지정된 제품 위험 항목 목록입니다. 이는 결과적으로, 각 위험을 충분한 테스트로 커버하기 위해 할당할 적절한 테스트 노력의 양을 결정하고, 수행할 테스트 작업의 효과와 효율성을 최적화하는 방식으로 이 테스트들의 순서를 정하는 데 도움이 될 것입니다. 추정된 작업들은 그 작업과 관련된 제품 위험 수준을 기반으로 우선순위가 지정될 수 있습니다. 위험이 더 높은 작업은 더 일찍 시작하고 더 많은 테스트 노력을 포함해야 합니다. 위험이 낮은 작업은 더 늦게 시작하고 더 적은 테스트 노력을 포함해야 합니다.

많은 조직에서 보안의 중요성을 고려하여, 위에 언급된 것 외에도 위협 모델링(threat modeling)을 사용하여 보안 요구 사항을 식별하고, 보안 위협 및 잠재적 취약점을 정확히 파악하며, 위협 및 취약점의 심각도를 정량화하고, 완화 방법을 우선순위화할 수 있습니다. 보안 위협은 심각도에 도달하기 위해 발생 가능성과 영향을 정량화하는 지침을 사용하여 우선순위가 지정됩니다.

반복 계획 동안 DevOps 팀은 다가오는 반복에서 구현될 사용자 스토리를 기반으로 제품 위험을 식별하고 분석합니다. 가급적 애자일 팀의 모든 구성원과 가능하면 일부 다른 이해관계자들이 제품 위험 세션에 참여합니다. 그 결과는 테스트를 위한 중요한 영역을 식별하는 우선순위가 지정된 제품 위험 항목 목록입니다. 이는 결과적으로, 각 위험을 충분한 테스트로 커버하기 위해 할당할 적절한 테스트 노력의 양을 결정하고, 수행할 테스트 작업의 효과와 효율성을 최적화하는 방식으로 이 테스트들의 순서를 정하는 데 도움이 될 것입니다.

추정된 작업들은 그 작업과 관련된 제품 위험 수준을 기반으로 우선순위가 지정될 수 있습니다. 위험이 더 높은 작업은 더 일찍 시작하고 더 많은 테스트 노력을 포함해야 합니다. 위험이 낮은 작업은 더 늦게 시작하고 더 적은 테스트 노력을 포함해야 합니다.

많은 조직에서 보안의 중요성을 고려하여, 위에 언급된 것 외에도 위험 모델링(threat modeling)을 사용하여 보안 요구 사항을 식별하고, 보안 위험 및 잠재적 취약점을 정확히 파악하며, 위험 및 취약점의 심각도를 정량화하고, 완화 방법을 우선순위화할 수 있습니다. 보안 위험은 각 위험의 발생 가능성과 영향을 정량화하는 지침을 사용하여 심각도에 도달하도록 우선순위가 지정됩니다.

2.2.2 SG1 테스트 접근 방식 수립 (Establish a Test Approach)

테스트 접근 방식은 식별되고 우선순위가 지정된 제품 위험을 완화하기 위해 정의됩니다. 특정 반복(iteration)에 대해 테스트할 항목과 기능은 반복 계획 동안 식별됩니다. 이 활동은 또한 제품 위험 세션의 결과를 기반으로 합니다. 테스트할 항목의 우선순위 목록은 일반적으로 해당 반복에서 테스트할 사용자 스토리와 관련됩니다. 기능은 일반적으로 테스트할 다양한 소프트웨어 품질 특성 등과 관련됩니다. 새로운 제품 위험이 반복 중에 명백해질 수 있으며, 이는 추가적인 테스트를 요구할 수 있습니다. 추가 테스트를 요구하는 새로운 제품 위험과 같은 문제들은 일반적으로 일일 스탠드업 회의에서 논의됩니다.

DevOps에서는 테스트 접근 방식이 CI/CD 파이프라인을 통한 지속적인 테스트(Continuous Testing)를 기반으로 정의됩니다. 지속적인 테스트는 전달 파이프라인의 각 단계에 적절한 실행 가능한 피드백과 자동화를 사용하여 일찍 그리고 자주 테스트하는 것을 의미합니다. 지속적인 통합(CI) 및 지속적인 배포(CD) 도구는 일반적으로 코드 커밋 시 자동으로 호출되며 빌드, 통합, 테스트, 보안, 규정 준수 및 배포 활동을 오케스트레이션합니다. CI/CD 파이프라인 내의 테스트 활동은 보통 애자일 테스트 사분면(Agile Testing Quadrants)을 사용하여 식별된 테스트 레벨을 기반으로 구축되며, 팀에 의해 지속적으로 최적화됩니다.

위험을 완화하기 위해 반복(iteration) 수준 또는 백로그 항목 수준에서 정의되는 테스트 접근 방식은 예를 들어 사용자 스토리 및 인수 기준에 대한 추가적인 검토, 위험 수준에 비례하는 테스트 노력, 위험 수준 및 유형에 기반한 적절한 테스트 기법 선택 등을 포함할 수 있습니다. 또한 배포 후에만 수행되는 테스트 활동을 제안할 수도 있는데, 예를 들어 사용자 경험에 대한 피드백을 수집하기 위해 프로덕션에서 수행되는 크라우드 테스트(crowd testing), 또는 다크 런치(dark launch)를 사용하여 실제 사용자 트래픽으로 성능을 테스트하는 것 등이 있습니다.

DevOps의 맥락에서 행동 주도 개발(BDD, Behavior-Driven Development)과 인수 테스트 주도 개발(ATDD, Acceptance Test-Driven Development)은 테스트 케이스의 식별 및 설계를 주도하는 데 인기 있는 기법입니다.

릴리스 수준의 테스트 접근 방식은 훨씬 더 높은 수준일 것이며 프로그램 또는 조직 수준에서 정의된 테스트 전략을 기반으로 해야 합니다. 테스트 접근 방식은 종종 팀/프로젝트 위키에 보관되거나 표시됩니다.

반복적인 개발에서 항상 명백한 중요한 위험은 회귀 위험(regression risk)입니다. 테스트 접근 방식은 회귀 위험을 어떻게 관리할지 정의해야 합니다. 일반적으로 이는 특정 회귀 테스트 세트를 생성함으로써 이루어지며, 이는 바람직하게는 자동화됩니다. 이 맥락에서 테스트 자동화 피라미드가 유용합니다. 이는 피라미드의 가장 낮은 수준에 있는 단위 테스트부터 시작하여 서비스 수준 테스트로 이동함으로써, 회귀 테스트 자동화의 가치를 극대화하는 방법을 보여줍니다. 사용자 인터페이스(UI) 테스트는 가장 상위에 위치합니다. 단위 테스트는 빠르고 신뢰할 수 있습니다. 서비스 계층은 사용자 인터페이스(UI)에 구애받지 않고 API 또는 서비스 수준에서 비즈니스 로직을 테스트할 수 있도록 합니다. 수준이 높을수록 테스트는 더 느려지고 더 취약해집니다.

자동화는 DevOps의 중요한 부분이므로, 테스트 자동화는 회귀 테스트 지원뿐만 아니라 DevOps 테스트 접근 방식의 주요 초점입니다. 회귀 위험을 다루는 또 다른 방법은 시스템의 관찰 가능성(observability)에 의존하여, 시스템의 상태와 사용을 지속적으로 모니터링하고, 예를 들어 이전의 안정적인 버전으로 롤백하는 자동 메커니즘을 구축하여 위험을 회피하는 것입니다. 진입 기준 (특정 프랙티스 2.3인 준비 완료 정의(Definition-of-Ready)라고도 함)은 일반적으로 정의된 테스트 접근 방식의 일부이지만, DevOps 개발에서는 다르게 처리될 가능성이 높습니다. DevOps 소프트웨어 개발 내에서 테스트는 팀 프로세스의 필수적인 부분이자 지속적인 활동입니다.

이전에 사용되던 진입 기준 체크리스트는 다음 두 가지로 나뉩니다:

1. 준비 완료 정의(Definition-of-Ready): 예를 들어, Given-When-Then 문을 사용하는 테스트 시나리오로 인수 기준(Acceptance Criteria)을 정의하는 것.
2. CI/CD 머신 내의 확인: 예를 들어, 풀 리퀘스트(pull request) 내의 확인, 더 큰 테스트 실행 활동을 시작하기 전에 새로 배포된 테스트 환경에서 스모크 테스트를 실행하는 것, 또는 필요한 테스트 데이터도 배포되었는지 확인하는 것.

결과적으로, 테스트를 시작할 수 있는지 여부를 결정하는 특정 체크리스트 또는 게이트웨이는 CI/CD 워크플로우에 구현되어야 하며, 다른 결함과 마찬가지로 부적합 시 빌드를 실패시켜야 합니다. 이는 DevOps 팀 내에서 한 테스트 단계(예: 단위 테스트)에서 다른 테스트 단계(예: 인수 테스트)로 구성 요소가 이동할 때도 적용됩니다. 테스트 종료 기준 (특정 프랙티스 2.4)은 CI/CD 파이프라인 내에서 달성되는 품질 게이트에 크게 의존하는 소위 완료 정의(Definition-of-Done, DoD)의 일부입니다. DoD는 예를 들어 테스트 커버리지 및 제품 품질과 같은 특정 테스트 관련 기준을 포함하는 것이 중요합니다. 반복은 합의된 사용자 스토리 세트의 구현으로 이어져야 하며, DoD에 정의된 (테스트) 종료 기준을 충족해야 합니다.

DevOps에서 "완료(DONE)"는 일반적으로 백로그 항목과 관련된 코드 변경 사항이 CI/CD 파이프라인 내의 모든 품질 게이트를 통과하여 프로덕션에 배포되었음을 의미합니다. 릴리스가 기능 토글(Feature Toggles)과 같은 기술을 사용하여 배포와 분리되어 요청 시 이루어지는 경우, 여러 반복에 걸쳐 적용될 수 있는 릴리스 수준의 DoD도 있습니다. 그러나 DevOps의 목표는 가능한 한 작은 덩어리(small chunks)로 자주 릴리스하는 것입니다. 릴리스 수준의 DoD는 다시 한번 일반적으로 커버리지 및 제품 품질 관련 기준을 포함하며, 주로 클라우드 테스트 또는 카나리 테스트와 같이 프로덕션에서 발생하는 테스트 활동을 기반으로 예상되는 가치가 사용자에게 전달되었는지 확인하는 데 중점을 둡니다.

특정 프랙티스 2.5인 중단 및 재개 기준 정의는 DevOps 라이프사이클에 관련성이 없을 가능성이 높습니다. 테스트는 DevOps 소프트웨어 개발 프로세스의 필수적인 부분이므로, 당연히 다른 반복 활동과 분리된 별도의 독립적인 활동으로 취급되지 않을 것입니다. 테스트 진행에 대한 잠재적 또는 실제 위협으로 간주될 수 있는 차단 문제가 있을 경우, 이는 일일 스탠드업 회의에서 논의됩니다. 이 회의에서 팀은 문제를 해결하기 위해 필요한 조치가 있는지, 있다면 어떤 조치를 취할지 결정할 것입니다. 따라서 공식적인 중단 및 재개 기준은 필요하지 않으며 정의되지 않을 것입니다. 이와 관련된 문제들은 일반적인 DevOps 루틴의 일부로 다루어집니다. 따라서 DevOps 루틴은 이 특정 프랙티스에 대한 대안적인 프랙티스 역할을 합니다.

2.2.3 SG3 테스트 추정치 수립 (Establish Test Estimates)

DevOps 팀의 경우, 테스트에 대한 상세 추정은 반복(iteration) 계획 동안 이루어질 것입니다. 상위 수준의 (테스트) 추정은 릴리스 계획 동안, 그리고 가능하면 백로그 다듬기 세션에서도 이루어집니다. 모든 추정은 당연히 각 스토리를 전달하는 데 필요한 모든 노력을 포함하는 팀 추정으로 수행됩니다.

추정 세션 동안 테스트 활동이 확실히 고려되도록 보장하는 것이 중요합니다. 이는 테스트 활동을 별도의 작업으로 식별하고 각 개별 테스트 작업을 추정하거나, 수행할 테스트가 명시되어 고려되도록 각 사용자 스토리를 추정함으로써 수행될 수 있습니다. DevOps에서는 거의 모든 것의 자동화가 예상되므로, 테스트 자동화, 테스트 환경 배포 자동화, 테스트 데이터 배포 자동화뿐만 아니라 해당 반복에 대해 식별된 테스트 접근 방식에 따른 파이프라인 개선을 포함하여 테스트 관련 활동의 자동화도 추정되어야 합니다. 추정 동안, DevOps 팀의 일원인 테스터는 추정 세션에 참여해야 합니다. 플래닝 포커(Planning Poker), 티셔츠 사이즈(T-shirt sizing), 스윔 레인 사이즈(swim lane sizing, 버킷 사이즈), 그리고 점수 투표(dot voting)는 모두 DevOps 소프트웨어 개발에서 사용되는 전형적인 추정 기법입니다.

다음 반복에 대해 수행될 작업은 일반적으로 사용자 스토리로 정의됩니다. 사용자 스토리는 추정 가능하려면 크기가 작아야 합니다. 추정 가능(Estimable)은 INVEST에 의해 정의된 사용자 스토리 기준 중 하나이며, 반복의 일부가 될 사용자 스토리에 적용 가능합니다. 반복 계획 동안 애자일 팀은 일반적으로 테스트 관련 작업을 식별하고 정의할 것입니다. 테스트 작업은 다른 개발 작업과 함께 작업 보드(task board)에 기록될 것입니다. 회고(retrospective) 시, 팀은 (문제로 식별된 경우) 추정 정확도를 되돌아보고 반복에서 수행된 개별 작업의 크기를 학습하여 개선해야 합니다.

릴리스 계획 시, 사용자 스토리 또는 에픽은 일반적으로 더 높은 수준으로 정의되며 아직 특정 작업으로 상세화되지 않았을 것입니다. 이는 당연히 추정을 더 어렵고 덜 정확하게 만들 것입니다. 언급된 바와 같이, DevOps 프로젝트는 추정을 위한 기초로 작업 분해 구조(work-breakdown structure)를 수립하지 않지만, 백로그 항목 우선순위 지정을 가능하게 할 만큼 충분한 추정 정확도를 충족하는 데 필요한 백로그 항목 개발 방법 및 내용에 대한 정보로부터 릴리스 수준에서 이점을 얻을 수 있습니다. DevOps의 목표가 가치 흐름을 최적화하는 것이므로, 추정은 무엇을 다음에 구축할지 결정하는 것을 지원해야 하며, 완전한 프로젝트 시간표를 만드는 것이 아닙니다.

DevOps 팀은 비교적 비공식적인 방식으로 추정하지만, 추정의 근거는 명확해야 합니다 (즉, 어떤 요소들이 고려되고 있는지). 근거에 기반한 논의는 추정의 더 높은 정확도를 촉진합니다. 일반적으로 DevOps 프로젝트에서 추정은 크기(스토리 포인트 사용) 또는 노력(이상적인 맨데이(man days)를 추정 단위로 사용)에 초점을 맞춥니다. 비용은 일반적으로 DevOps 프로젝트의 추정 세션에서 다루어지지 않습니다. 인프라 비용은 DevOps 환경에서 일반적으로 높으며, 이러한 비용 추정치는 일반적으로 팀 추정 세션이 아닌 특정 추정 세션에서 수립됩니다. 공용 클라우드 사용과 같은 인프라 비용에 대한 면밀한 모니터링 및 통제가 필요하며, 자동 보고와 함께 CI/CD 파이프라인에 내장될 수 있습니다. (예를 들어, 실행된 모든 GitHub Action 작업은 사용된 vCPU 분으로 제시됩니다) .

따라서 이 TMMi의 구체적인 목표와 그 특정 프랙티스는 특정 프랙티스 3.2 테스트 라이프사이클 정의를 제외하고 완전히 적용 가능합니다. 반복적 및 DevOps 소프트웨어 개발의 기본 사항 중 하나는 작은 덩어리로 작업하는 것입니다. 따라서 식별된 작업은 일반적으로 (테스트) 추정의 기초로 사용되기에 충분히 상세합니다. 따라서 DevOps 환경에서는 추정을 위한 추가적인 기초로 삼기 위해 테스트 활동을 위한 라이프사이클을 정의할 필요가 없습니다.

2.2.4 SG4 테스트 계획 개발 (Develop a Test Plan)

테스트 계획은 사전 사고("활동")에 관한 것이지, 연관된 테스트 계획서("문서")를 정의하는 것에 관한 것이 아니라는 점을 다시 한번 강조합니다. DevOps 에서 이 TMMi 의 구체적인 목표에 의해 정의된 테스트 계획 활동의 대부분은 릴리스 및 반복 계획뿐만 아니라 백로그 항목 다듬기 세션에서도 수행될 것입니다.

그러나 이러한 활동의 결과는 일반적으로 테스트 계획에 문서화되지 않을 것이며, 특히 반복 계획의 경우 작업 보드(task board)에 반영될 수 있습니다.

테스팅은 릴리스 및 반복 계획의 필수적인 부분으로 수행되므로, 그 결과인 "일정"에는 테스트 활동도 포함될 것입니다. 순차적 라이프사이클에서 개발되는 상세한 일정보다는, DevOps 프로젝트 내의 일정은 릴리스 및 반복 우선순위를 반영하는 사용자 스토리(백로그 항목) 및 작업의 순서에 더 가깝습니다. 이는 예를 들어 원하는 비즈니스 가치 전달에 기반합니다. 테스트 실행은 주로 코드 커밋 시 트리거되는 CI/CD 파이프라인 내에서 발생한다는 점을 명심하십시오. 추가적인 수동 테스트이나 프로덕션에서 수행되는 활동은 반복 보드에 계획될 수 있습니다. 마인드맵이 여기서 지원 기법으로 자주 사용됩니다. 작업 보드는 반복 우선순위를 반영할 것입니다. 따라서 명시적인 (테스트) 일정이 수립되지는 않으며, 대신 사용자 스토리에 대해, 그리고 수행할 작업(테스팅 작업 포함)에 대해 명확한 릴리스 및 반복 우선순위가 정의될 것으로 예상됩니다.

프로젝트 수준에서 테스트 인력 배치(staffing)는 팀 구축의 일부입니다. 테스트 또는 다기능(multi-skilled) 리소스의 필요성은 사전에 식별됩니다. 프로젝트가 변경되거나 성장함에 따라, 개인을 특정 팀/프로젝트에 처음 선택한 근거를 잊어버리기 쉬운데, 여기에는 종종 팀/프로젝트와 특별히 관련된 특정 기술 요구 사항이나 경험이 포함됩니다. 이는 사람들의 기술 요구 사항을 기록하여, 그들이 팀/프로젝트에 선택된 이유와 관련된 백업 정보를 제공하는 좋은 근거가 됩니다.

일단 DevOps 팀이 정의되면 테스트 인력 배치는 거의 고정됩니다. 반복 계획 동안, 필요한 경우 팀이 필요한 테스트를 수행하기에 충분한 테스트 리소스, 지식 및 기술을 가지고 있는지 확인하기 위해, 예를 들어 비기능 테스트를 위한 (추가적인) 리소스와 기술의 식별이 논의될 수 있습니다.

비록 일반적으로 구체적이고 상세한 테스트 계획은 개발 및 문서화되지 않지만, 특정 프랙티스 4.5 테스트 계획 수립은 DevOps 환경 내에서 여전히 관련성이 있습니다. 그러나 테스트 계획의 맥락에서 발생하는 논의의 결과는 일반적으로 간결한 형태로, 아마도 마인드맵이나 단 한 페이지의 문서 (그림 4 참조), 팀 수준의 애자일 테스트 사분면(Agile Testing Quadrants), 그리고 CI/CD 파이프라인 내의 테스트 활동으로 기록될 가능성이 높습니다.

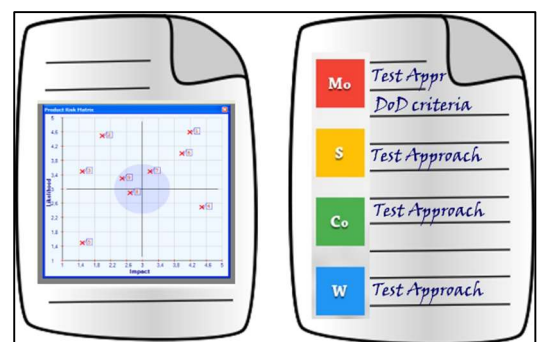


Figure 4: A One Page Test Plan

2.2.5 SG5 테스트 계획에 대한 약속 확보 (Obtain commitment to the Test Plan)

DevOps 내에서 테스트 접근 방식과 테스트 계획을 개발하고 수립하는 프로세스는 팀 기반의 활동이며, 아마도 (팀 구성원 중 한 명인) 테스트 전문가가 이끌 것입니다. 제품 품질은 팀의 책임입니다. 따라서 팀이 올바른

프로세스를 따르고 합의된 테스트 활동이 CI/CD 파이프라인에 내장되어 있다면, (테스트) 접근 방식과 (테스트) 계획에 대한 약속은 팀 활동의 결과로서 암묵적으로 이루어진 것으로 간주됩니다. 이는 테스터가 테스트 계획을 준비하고 난 후 명시적인 약속을 확보해야 하는 전통적인 환경에서의 작업 방식과는 큰 차이입니다.

테스트 계획을 구현하기 위해 필요한, 배포된 테스트 환경 및 실행된 자동 테스트 형태의 인프라 리소스는 고려되어야 하며 이해관계자에 의해 제공되어야 합니다. DevOps 환경에서 팀(제품 책임자 포함)은 우선순위가 지정된 제품 위험 목록과 수행할 테스트 완화 조치에 대해 이해하고 동의해야 합니다. 이러한 이해와 약속은 예를 들어, 릴리스 또는 반복 계획 동안 제품 위험, 테스트 접근 방식 및 그 근거를 팀에게 설명하는 짧은 발표와 그에 이은 논의를 통해 달성될 수 있습니다.

추정 세션 (SG 3 테스트 추정치 수립 참조) 동안 작업량은 추정됩니다. 팀을 위한 (테스트) 리소스는 DevOps 팀의 구성 설정에 의해 고정됩니다. 추정되고 개발 대상으로 선택된 사용자 스토리는 가용한 리소스를 시작점(제약 조건)으로 삼습니다. 따라서 다시 한번, 작업과 리소스 수준을 조정하는 것은 의미 있는 활동이 아니며, 그러나 클라우드 컴퓨팅/스토리지/네트워크 가용성과 같은 기술 리소스에 지속적으로 집중하는 것이 핵심이며 팀에게 제공되어야 합니다. 따라서 특정 프랙티스 3.2 작업과 리소스 수준 조정은 DevOps 환경에서는 일반적으로 관련성이 없는 프랙티스입니다. 일일 스탠드업 회의와 스크럼 오브 스크럼(Scrum-of-Scrum) 회의는 반복 동안 발생하는 모든 리소스 문제를 다루고, 적절한 리소스를 즉시 (재)할당하거나, 전달 가능한 항목을 향후 반복 또는 릴리스 계획에서 조정되도록 현재 반복에서 제거하는 데 사용될 것입니다.

2.3 프로세스 영역 2.3 테스트 모니터링 및 통제 (Test Monitoring and Control)

테스트 모니터링 및 통제의 목적은 테스트 진행 상황이 계획에서 크게 벗어나거나 제품 품질이 기대치에서 크게 벗어날 때 적절한 시정 조치를 취할 수 있도록 테스트 진행 상황 및 제품 품질에 대한 이해를 제공하는 것입니다.

DevOps 환경에서의 모니터링 및 통제는 전통적인 프로젝트와 비교할 때 근본적으로 다른 몇 가지 사항이 있습니다. 모니터링 및 통제가 DevOps 내에서 필수적인 요소이긴 하지만, 이것이 경직된 계획을 고수하는 것이 목표임을 의미하지는 않습니다. 테스트 모니터링 및 통제 관점에서 볼 때, 이는 우리가 계획 주도적이지 않고, 테스트로부터 얻은 진행 상황과 결과를 끊임없이 검토하며, 예를 들어 새로운 제품 위험이 명백해짐에 따라 계획과 접근 방식을 적절하게 조정한다는 것을 의미합니다.

일반적으로 잘 다듬어진 인수 기준, 백로그 항목 (위험 수준 포함), 그리고 작업 등으로 분산되어 있는 테스트 계획은 살아있는 실체(living entity)이며, 새로운 정보가 이용 가능해지거나 피드백이 주어짐에 따라 지속적으로 검토되고 업데이트될 필요가 있습니다. 테스트 활동의 목표는 주어진 백로그 항목에 대한 인수 기준 세트, 주어진 백로그 항목에 대해 업데이트된 팀의 테스트 접근 방식, 그리고 백로그 수준에서 정의된 특정 완료 정의(Definition-of-Done) 기준에 의해 정의됩니다. DevOps 프로젝트는 또한 '더 큰 그림'을 염두에 두고 반복 수준뿐만 아니라 CI/CD 수준에서도 모니터링하고 통제해야 합니다.

테스팅은 DevOps 팀의 전반적인 프로세스에 완전히 통합된 프로세스이므로, 테스트 모니터링 및 통제 또한 DevOps 팀의 전반적인 모니터링 및 통제 메커니즘의 필수적인 부분이라는 점에 유의하는 것이 중요합니다. 그 결과, 테스터들은 전통적인 프로젝트에서처럼 테스트 관리자에게 보고하는 것이 아니라, 팀에게 보고합니다.

2.3.1 SG1 계획 대비 테스트 진행 상황 모니터링 (Monitor Test Progress against Plan)

DevOps 팀은 테스트 진행 상황을 모니터링하고 기록하기 위해 다양한 방법과 도구를 활용합니다. 예를 들어, 애자일 작업 보드(Agile task board) 상의 테스트 작업 및 스토리 진행 상황, 그리고 번다운 차트(burndown charts) 등이 있습니다. 이러한 정보는 위키 대시보드 및 대시보드 스타일의 이메일과 같은 매체를 사용하거나, 스탠드업 회의 중에 구두로 팀의 나머지 구성원에게 전달될 수 있습니다. 팀은 전체 릴리스 및 각 반복(iteration) 내의 진행 상황을 추적하기 위해 번다운 차트를 모두 사용할 수 있습니다. 번다운 차트는 일반적으로 예상 속도(velocity)와 구현할 기능 백로그에 대한 진행 상황을 보여줍니다 (따라서 팀의 성과를 보여줍니다).

또한, CI/CD 파이프라인 전반에 걸쳐 인프라, 애플리케이션 및 네트워크를 자동화된 방식으로 모니터링하기 위해 온라인 대시보드를 사용한 지속적인 모니터링이 DevOps 에서 사용될 것입니다. 테스트 환경 리소스는 반복 계획 동안 합의된 리소스 및 CI/CD 파이프라인 사용량에 대비하여 지속적으로 모니터링됩니다.

또 다른 일반적인 프랙티스는 작업 보드를 통해 테스트 환경 문제를 추적하는 것입니다. 예를 들어, 스토리 카드에 '테스트 환경 차단됨(blocked test environment)' 스티커를 사용하거나, 환경으로 인해 차단된 모든 스토리가 차단 해제될 때까지 기다리는 별도의 열을 보드에 생성하는 방식입니다. 이는 진행을 막는 테스트 환경의 영향에 대한 가시성을 확보하고, 팀이 다른 작업을 수행하기 전에 이 문제를 해결하는 데 주의를 집중하게 만드는 것입니다.

팀과 CI/CD 의 현재 상태(테스팅 상태 포함)를 즉각적이고 상세하게 시각적으로 나타내기 위해, 팀은 일반적으로 작업 보드를 사용합니다. 반복 계획 동안 생성된 스토리 카드, 개발 작업, 테스트 작업 및 기타 작업은 작업 보드에 기록됩니다. 반복 동안, 진행 상황은 이러한 작업이 "할 일(to do)", "진행 중(work in progress)", "완료(done)"와 같은 열을 가로질러 이동함에 따라 관리됩니다. 흐름을 유지하기 위해, 작업 진행 중 한계(WIP limits)가 칸반 보드의 다른 상태에 적용되어, 팀이 새로운 작업을 시작하는 대신 (진행 중인) 작업을 완료하도록 행동을 통제하는 경우가 많습니다. DevOps 팀은 일반적으로 스토리 카드를 작업 보드에 유지하기 위해 도구를 사용하며, 이는 대시보드와 상태 개요를 자동화할 수 있습니다.

그러나 일부 팀은 개별 활동에 대한 특정 작업을 생성하지 않고, 스토리 카드만을 사용하며, 테스팅이 문서화될 수 있는 도구나 위키를 참조하는 주석을 이 카드에 달 수도 있습니다. 작업 보드의 테스팅 작업은 일반적으로 사용자 스토리에 대해 정의된 인수 기준과 모든 백로그 항목에 대한 제약 조건인 비기능 요구 사항과 관련됩니다. 테스트 자동화 스크립트와 테스트 작업에 대한 탐색적 테스트가 통과 상태를 달성하면, 해당 작업은 작업 보드의 "완료" 열로 이동합니다.

완료 정의(Definition-of-Done, DoD)는 진행 상황이 측정되는 종료 기준 역할을 합니다. DoD 는 또한 테스팅 활동과 관련되어야 하며, 사용자 스토리의 개발 및 테스팅이 '완료'로 불리기 전에 충족되어야 하는 모든 기준을 보여줍니다. 테스트 작업과 관련된 테스팅 기준은 팀이 완료하기로 합의한 완료 정의의 일부만을 구성한다는 점에 유의해야 합니다. 완료 정의 기준은 일반적으로 스토리 수준 및 에픽 수준과 같은 다중 레벨에 적용됩니다. 팀 전체는 작업 보드의 상태를 정기적으로 검토하며, 종종 일일 스탠드업 회의 중에, 작업이 허용 가능한 속도로 보드를 가로질러 이동하고 있는지 확인합니다. 만약 어떤 작업(테스팅 작업 포함)이 움직이지 않거나 너무 느리게 움직인다면, 이는 해당 작업의 진행을 막는 문제를 분석하는 팀 기반의 논의를 촉발합니다. 일일 스탠드업 회의에는 테스터를 포함한 DevOps 팀의 모든 구성원이 참여합니다. 이 회의에서 그들은 현재 상태와 실제 진행 상황을 팀의 나머지 구성원에게 전달합니다. 개발 또는 테스트 진행을 막을 수 있는 모든 장애물은

일일 스탠드업 회의 중에 전달되어, 팀 전체가 이를 인지하고 그에 따라 조치할 수 있게 됩니다. 이러한 방식으로 프로젝트 위험 관리가 이러한 일일 회의에 통합됩니다. 정기적으로 팀은 프로젝트 위험의 상태 모니터링을 지원하기 위해 ROAMing 연습을 수행하기로 결정할 수 있습니다. 테스트 환경 부족과 같은 테스트에 대한 위험을 포함하여 모든 프로젝트 위험은 일일 스탠드업 중에 전달되고 해결될 수 있습니다. (참고로, 제품 위험 모니터링은 제품 품질 모니터링의 일부이므로, 이후 특정 목표 SG 2 계획 및 기대치 대비 제품 품질 모니터링과 함께 논의됩니다.)

일일 작업 관리를 위한 일일 스탠드업 회의와 작업 경로 수정을 위한 DevOps 팀 프랙티스는 테스트 모니터링 및 통제 프로세스 영역의 TMMi 특정 프랙티스의 의도를 충족하는 검증된 좋은 기법입니다. 반복이 끝날 때, 합의된 반복 목표에 대비하여 스프린트 검토(sprint review)가 개최됩니다. 팀은 달성된 사항, 예를 들어 어떤 스토리 또는 에픽이 완료되었는지를 시연합니다. CI/CD 파이프라인을 통한 통합은 당연히 지속적입니다. 완료 정의(Definition-of-Done) 등에 대비한 테스트의 성과는 반복 검토의 일부가 될 것입니다. 이해관계자들과 함께 시연(Demo)이 조직되어 전달되는 제품의 비즈니스 가치와 품질에 대해 논의합니다.

이해관계자는 반복 계획, 반복 검토(시연) 및 회고(retrospectives)에서 제품 책임자(product owner)에 의해 대표됩니다. 제품 책임자는 제품 백로그에 대한 논의를 통해 참여하며, 반복 전반에 걸쳐 사용자 스토리 상세화 및 테스트 설계에 대한 피드백과 입력을 제공합니다. 다른 이해관계자들은 반복 검토 동안 각 반복이 끝날 때 참여합니다. 이해관계자의 대표가 애자일 작업 방식에 내재되어 있으므로, 이해관계자 참여에 대한 특정 모니터링은 요구되지 않습니다.

2.3.2 SG2 계획 및 기대치 대비 제품 품질 모니터링 (Monitor Product Quality against Plan and Expectations)

제품 품질 모니터링을 위해서는 진행 상황 모니터링과 거의 동일한 메커니즘이 사용됩니다 (위의 SG1 참조). 온라인 대시보드를 사용한 제품 품질 모니터링은 팀이 개발 및 테스트 중에 수행하는 것 외에도, 소프트웨어 제품의 품질이 프로덕션에서 모니터링되는 지속적인 프로세스이기도 합니다. 운영(Ops) 역할은 관찰 가능성(observability)을 사용하여, 일반적으로 서비스 수준 지표(SLI)로 정의된 것을 통해 제품의 품질을 지속적으로 모니터링합니다. SLI의 회귀 또는 편차는 품질 및 사용자 경험을 보존하기 위한 시정 조치를 촉발합니다.

DevOps에서 제품 위험 모니터링의 초점은 상세한 위험 문서보다는 정기적인 회의에서 제품 위험 목록, 보안 위험/위험 등을 검토하는 데 있습니다. 새로 식별되거나 변경된 제품 위험(예: 탐색적 테스트의 결과)은 논의되고, 필요한 테스트 조치가 합의될 것입니다. 필요할 경우, 추가적인 ROAMing 연습이 요구될 수 있습니다. 다양한 제품 위험의 상태는 일반적으로 대시보드의 차트를 통해 표시됩니다.

DevOps 팀은 제품 품질을 모니터링하고 개선하기 위해 테스트 합격/불합격을, 결함 발견율, 발견 및 수정된 결함 등 전통적인 개발 방법론에서 수집되는 것과 유사한 결함 기반 메트릭을 사용합니다. 반복 동안 발견 및 해결된 결함 수와, 다음 반복의 백로그 일부가 될 가능성이 있는 미해결 결함 수는 일일 스탠드업 회의 동안 모니터링되어야 합니다. 전반적인 제품 품질을 모니터링하고 개선하기 위해, 많은 DevOps 팀은 제품이 고객 기대치를 충족하는지 여부에 대한 피드백을 받기 위해 고객 만족도 조사도 사용합니다.

테스트 커버리지 및 제품 품질에 대한 테스트 종료 기준은 완료 정의(Definition-of-Done, DoD)의 일부입니다. 합의된 종료 기준의 준수는 일반적으로 작업 보드를 통해 모니터링되며, 백로그 항목이 (주로 해당 백로그 항목이 배포되었을 때—배포된 후에야 "완료"임) DoD 기준을 준수하는 경우에만 스토리 카드가 "완료"로 표시될 수 있습니다. 지속적인 테스트를 위해 특정 종료 기준이 정의되는 것이 일반적입니다. 결함은 CI/CD 파이프라인 내의 지속적인 테스트의 일부로 지속적으로 모니터링됩니다. 프로덕션에서 제품의 예상 품질 특성 충족 여부는 관찰 가능성을 사용하여 모니터링됩니다. 제품 품질이 기대치를 충족하지 못할 때 배포를 차단하기 위해 자동화된 품질 게이트 (정의된 임계값 기반)가 DevOps 파이프라인의 여러 단계에 구현되는 경우가 많습니다. 이러한 게이트는 테스트 합격률, 성능 벤치마크, 코드 커버리지, 정적 코드 분석 결과와 같은 메트릭을 평가할 수 있습니다.

성능 및 신뢰성은 종종 중요한 비기능적 품질 특성이며, 따라서 제품 품질 모니터링의 일부입니다. 성능 테스트(예: 부하 및 스트레스 테스트)은 CI/CD 파이프라인에 통합되어 시스템 성능을 예상 벤치마크와 비교하여 지속적으로 모니터링할 수 있습니다. 성능 임계값이 위반될 경우 경고가 트리거되어 팀에게 즉시 문제를 해결하도록 알립니다. 또한 (로그) 모니터링 도구를 사용하여 프로덕션에서 실시간 오류율, 메모리 누수 및 시스템 충돌을 추적할 수 있습니다. 이러한 문제들을 조기에 식별하는 것은 제품이 신뢰성 기대치를 충족하도록 보장하는 데 도움이 될 수 있습니다.

일일 스탠드업 회의는 거의 지속적으로 제품 품질 검토를 수행하는 데 사용되는 메커니즘입니다. 이해관계자들과 함께 시연이 조직되어 전달되는 제품의 비즈니스 가치와 품질에 대해 논의합니다. 제품 품질은 정의된 DoD 품질 기준에 대비하여 검증 및 확인됩니다.

테스트 계획 프로세스 영역에 따라, 진입, 중단 및 재개 기준 모니터링에 대한 특정 프랙티스는 관련성이 없을 가능성이 높습니다. 이 기준들이 DevOps 환경에서 일반적으로 관련성이 없는 이유에 대한 설명은 테스트 계획 프로세스 영역의 특정 목표 2 테스트 접근 방식 수립을 참조하십시오.

2.3.3 SG3 시정 조치 종결 관리 (Manage Corrective Actions to Closure)

DevOps 팀은 번다운 차트에서 기대치와의 편차 또는 작업 보드 상의 (테스트) 작업 및 스토리 진행 부족과 같은 문제들을 매우 신속하게 알아차릴 것입니다. 가장 중요한 것은, 프로덕션의 소프트웨어 품질이 저하되고 있다면 즉시 인지되고, 일반적으로 알려진 안정적인 버전으로 롤백하는 것과 같은 시정 조치를 촉발한다는 것입니다.

이러한 문제들(예: 테스트 진행을 막는 문제)은 일일 스탠드업 회의 중에 전달되어 팀 전체가 이를 인지하게 됩니다. 이는 문제를 분석하는 팀 기반의 논의를 촉발합니다. 팀은 제품 책임자와 함께 취해야 할 시정 조치를 결정할 것입니다.

DevOps 기반 프로젝트에서 시정 조치 관리의 주요 자율 조직 팀(self-organizing team)의 책임입니다. 팀은 적절한 시정 조치를 정의하고 구현하거나, 모든 문제를 스크럼 마스터에게 '장애물(impediments)'로 에스컬레이션할 수 있습니다. 제품 품질에 대한 시정 조치는 때때로 자동화되며, 그 재발을 방지하기 위해 근본 원인이 모니터링됩니다. 이는 DevOps 팀의 지속적인 개선 문화의 일부입니다.

스크럼 마스터는 일반적으로 팀이 문제를 종결할 때까지 지원하는 책임이 있습니다. 시정 조치를 논의하고 관리하는 전형적인 이벤트는 일일 스탠드업 회의와 회고 회의(retrospective meetings)입니다. 합의된 시정

조치는 제품 백로그를 통해 '작업' 또는 '백로그 항목'으로 관리되거나 (반복 내에서는) 작업 보드를 통해 종결될 때까지 관리될 수 있습니다.

2.4 프로세스 영역 2.4 테스트 설계 및 실행 (Test Design and Execution)

테스트 설계 및 실행 프로세스 영역의 목적은 테스트 설계 명세서 수립, 테스트 설계 기법 사용, 구조화된 테스트 실행 프로세스 수행, 그리고 테스트 인시던트 종결 관리를 통해 테스트 설계 및 실행 중 테스트 프로세스 역량을 개선하는 것입니다.

근본적인 목표("위험을 완화하고 소프트웨어를 테스트하는 것")는 DevOps 프로젝트와 전통적인 순차적 프로젝트에서 동일하지만, 테스트를 수행하는 접근 방식은 일반적으로 매우 다릅니다. DevOps에서는 유연성과 변화에 대응할 수 있는 능력이 중요한 출발점입니다. 또한, 테스트 분석 및 설계, 테스트 구현, 테스트 실행은 순차적인 테스트 단계가 아니라 빠른 피드백 주기의 지원을 받아 병렬적으로, 중복되며, 반복적으로 수행됩니다.

수립되는 테스트 문서의 세부 수준은 또 다른 주요 차이점이며, 주로 테스트 케이스 자동화 코드의 일부로서 문서화에 의존합니다. 수동 테스트의 경우, DevOps 프로젝트에서는 테스트 자동화를 보완하기 위해 일반적으로 경험 기반(experienced-based) 및 결함 기반(defect-based) 기법이 사용됩니다. 또한, 소프트웨어를 프로덕션에 배포한 후, 그리고 미리 선택된 사람들에게 릴리스한 후, 사용자들이 일부 지침/테스트 헌장(test charters)을 따르고 유용성 또는 접근성과 같은 전용 품질 특성에 대한 추가 피드백을 개발팀에 제공하는 크라우드 테스트(crowd testing)에서 수동 테스트가 사용될 수 있습니다.

명세 기반(Specification-based) 테스트 설계 기법도 여전히 적용 가능하며 사용되는데, 특히 협업적 요구 사항 분석의 일부로 사용자 스토리 인수 기준을 향상시키기 위한 가능한 예시 시나리오를 나열할 때 그러합니다. 단위 테스트 및 통합 테스트에 더 중점을 두는 따라, 구문(statement) 및 결정(decision) 테스트와 같은 화이트 박스(white-box) 기법뿐만 아니라 API에 대한 계약 테스트(contract testing)도 훨씬 더 인기가 있습니다.

마지막 주요 차이점은 높은 변경율로 인해 발생하는 회귀 위험 수준이며, 이는 다양한 테스트 레벨에서 더 많은 회귀 테스트를 요구합니다. 이상적으로, 회귀 테스트는 높은 수준으로 자동화되어야 하며, CI/CD 파이프라인 내의 여러 테스트 단계에서 자동으로 트리거 및 실행되어야 합니다.

많은 차이점들이 있지만, 궁극적으로 테스트 설계 및 실행 프로세스 영역의 맥락에서 이는 항상 테스트를 생성하고, 제품 위험을 완화하며, 테스트를 실행하고 결함을 찾는 것에 관한 것입니다.

2.4.1 SG1 테스트 설계 기법을 이용한 테스트 분석 및 설계 수행 (Perform Test Analysis and Design using Test Design Techniques)

DevOps에서 테스트 분석 및 설계, 그리고 테스트 실행은 일반적으로 전체 제품 팀의 애자일 작업 방식에 내장되어 지속적으로 상호 지원하며 실행되는 활동입니다. DevOps 프로젝트에서 테스터는 협업하여 사용자 스토리를 생성하고 다듬는 팀의 일부입니다. 요구 사항이 개발되는 동안 빈번한 비공식 검토가 수행되며, 여기에는 각 사용자 스토리에 대한 인수 기준이 포함됩니다. 이러한 기준은 비즈니스 담당자, 개발자, 테스터 간의 협업을 통해 정의됩니다. 일반적으로 테스터의 고유한 관점은 누락된 세부 사항, 대안 시나리오를 식별하고 스토리를 테스트 가능하게 만듦으로써 사용자 스토리를 개선할 것입니다. 따라서 테스트 분석은

명시적인 별도의 활동이 아니라, 테스터가 협업적인 사용자 스토리 개발 역할의 일부로서 수행하는 암묵적인 활동입니다.

사용자 스토리 분석을 기반으로 테스트 조건이 식별됩니다. 테스트 관점에서 테스트 기반은 무엇을 테스트할 수 있는지 확인하기 위해 분석되며, 이것이 테스트 조건이 됩니다. 정의된 인수 기준이 충분히 상세하고 명확하다면, 이는 전통적인 테스트 조건의 역할을 대신할 수 있습니다. DevOps 환경에서, 종종 Gherkin 구문을 사용하는 행동 주도 개발(BDD) 및 인수 테스트 주도 개발(ATDD)과 같은 기법이 시나리오를 정의하고 테스트 조건을 식별하는 데 도움이 될 수 있으며, 이를 주어진 사용자 스토리의 인수 기준으로 문서화합니다. 마찬가지로, 비기능 요구 사항에 대해서도 특정 테스트 조건이 인수 기준 형식으로 자주 정의됩니다.

테스트 조건은 이후 테스트로 변환되며, 이는 정의되고 합의된 인수 기준을 커버하기 위해 필요합니다. 흥미로운 테스트 조건은 탐색적 테스트에 사용될 테스트 헌장(test charters)에도 기록될 수 있습니다. 물론 위 내용은 스토리 수준뿐만 아니라 에픽 및 기능 수준의 테스트에도 동일하게 적용됩니다. 에픽 및 기능 수준의 테스트 조건은 일반적으로 사용자 스토리 수준의 테스트 조건보다 더 추상적이며 여러 사용자 스토리에 걸쳐 있거나 비기능 요구 사항을 나타냅니다.

명세 기반 테스트 설계 기법은 사용자 스토리와 인수 기준으로부터 테스트 조건을 도출하는 데 일반적으로 유용합니다. 그러나 DevOps 환경에서는 테스터가 경험을 바탕으로 해당 기법을 숙달하여 상황에 따라 유연하게 사용할 수 있으므로, 이러한 테스트 설계 기법들이 명시적으로 사용되기보다는 암묵적으로 사용되는 경우가 훨씬 많습니다.

DevOps 에서 테스트 우선(test-first) 원칙이 적용됨에 따라, 일련의 테스트 조건을 커버하는 테스트는 코드 개발보다 이전 또는 적어도 병렬로 식별되고 (그리고 가능하면 자동화됩니다). 이러한 접근 방식은 때때로 백로그 다듬기에서 이미 시작됩니다. 자동화된 단위 테스트의 경우 테스트 주도 개발(TDD)과 같은 접근 방식이 고려될 수 있습니다. 더 높은 테스트 레벨의 경우, 이미 언급했듯이 BDD 와 ATDD 는 인기 있는 접근 방식이며 테스트 자동화와도 밀접하게 관련되어 있습니다. 대부분의 수동 테스트의 경우, 팀이 테스트 실행을 진행함에 따라 테스트가 식별/다듬어질 것입니다. 테스트는 전통적인 프로젝트에서처럼 상세하게 문서화되지 않는 경우가 많으며, 오히려 탐색적 테스트를 사용하여 테스트 아이디어 형식으로 문서화됩니다. 복잡하고 중요한 영역의 경우, 보다 전통적인 테스트 설계 및 테스트 케이스 개발 접근 방식(공식 테스트 설계 기법 사용)이 위험을 커버하는 최선의 방법일 수 있습니다. 그러나 이 접근 방식에서도 전통적인 순차적 라이프사이클 환경에서의 테스트와 비교할 때 문서의 양은 제한될 것입니다.

테스트의 우선순위는 해당 테스트가 커버하는 사용자 스토리의 우선순위를 따릅니다. 사용자 스토리의 우선순위는 비즈니스 가치를 기반으로 하며, 가장 높은 우선순위는 가장 높은 비즈니스 가치와 관련됩니다. 테스트는 기능적 및 비기능적 위험을 커버하기 위해 수립되는 것이 중요하며, 특히 DevOps 에서는 회귀 위험을 커버하는 것이 중요합니다. 테스트 우선순위는 테스트를 CI/CD 파이프라인 작업에 추가함으로써 구현됩니다. 테스트 조건 및 테스트 실행을 지원하는 데 필요한 특정 테스트 데이터가 식별됩니다. 그러나 DevOps 에서는 전통적인 환경과 달리, 필요한 테스트 데이터가 일반적으로 테스트 명세 문서의 일부로 먼저 명세되지 않습니다. 테스트 데이터는 오히려 필요한 도구 및/또는 기능이 제공되는 즉시 제공됩니다. 수동 테스트의 거의 즉각적인 실행을 허용하기 위해 테스트 데이터가 즉시 생성됩니다. 그러나 자동화된 테스트의 경우 데이터는 일반적으로 사전에 명세되어야 할 것입니다. 테스트 데이터는 종종 테스트 환경과 동일한 방식으로 관리되며, 요청 시

이러한 환경에 배포될 수 있습니다. 테스트 데이터 준비 도구는 개인 정보 보호 및 GDPR 규정에 따라 필요에 따라 프로덕션 데이터를 익명화, 무작위화 및 마스킹할 수 있습니다.

요구 사항, 테스트 조건 및 테스트 간의 추적성은 수립되고 유지되어야 합니다. 팀은 테스트의 일부로서 다양한 사용자 스토리와 인수 기준을 커버했음을 명확히 해야 합니다. 따라서 팀은 각 사용자 스토리에 식별자를 할당하고, 이 식별자를 사용하여 합의된 기준에 따라 테스트가 완료되었는지 확인하는 것을 지원하도록 요구 사항(사용자 스토리)을 조직하고 관리할 도구 지원이 필요합니다. 일반적으로 도구 내에서 연결된 테스트와 사용자 스토리는 실시간 커버리지 측정 가능성을 제공합니다. 테스트-스토리 추적성은 CI/CD 파이프라인 테스트 작업 상태 및 로그를 통해 자동으로 생성되어 가시화됩니다.

2.4.2 SG2 테스트 구현 수행 (Perform Test Implementation)

테스트 구현은 테스트 실행을 시작하는 데 필요한 모든 것을 제자리에 배치하는 것에 관한 것입니다. 일반적으로 테스트 실행을 지원하기 위한 테스트 문서 개발은 최소화됩니다. 대신, 자동화된 (회귀) 테스트 스크립트가 개발되고 우선순위가 지정됩니다. 회귀 테스트 준비는 가능한 한 빨리 다른 테스트 활동과 병렬로 시작됩니다.

DevOps 환경에서는 수동 테스트를 위한 상세한 테스트 절차는 일반적인 프랙티스가 아닙니다. 지식이 풍부한 팀 내에서 작업할 때, 테스트는 훨씬 더 높은 수준의 추상화로 문서화될 가능성이 높습니다. 예를 들어, 탐색적 테스트(exploratory testing)를 위한 테스트 헌장(test charters)을 사용하는 방식입니다. 테스트를 실행하는 사람들이 필요한 도메인 및 기술 지식을 가지고 있을 것으로 예상되므로, 이는 테스트를 실행하는 사람들에게 충분할 것입니다. 테스트를 실행하는 사람들은 팀 내에서 일하기 때문에, 코딩된 내용과 기능의 목적 적합성(fit for purpose) 모두에 대해 더 잘 이해할 것입니다. 반복 간 및 반복 내의 변경 수준이 일반적으로 높기 때문에, 상세한 테스트 절차를 개발하는 것은 또한 많은 유지보수 작업을 초래할 것입니다.

일반적으로 많은 수동 테스트는 탐색적 테스트를 사용하여 수행됩니다. 탐색적 테스트에서는 상세한 테스트 절차가 개발되지 않고, 대신 테스트 아이디어와 테스터를 위한 지침을 설명하는 상위 수준의 한 페이지짜리 테스트 헌장이 개발될 것입니다.

그러나 대부분의 DevOps 팀은 자동화된 테스트를 사용합니다. 사용되는 전형적인 접근 방식은 TDD(Test-Driven Development), BDD(Behavior-Driven Development), ATDD(Acceptance Test-Driven Development)입니다. BDD 자체는 테스트 자동화는 아니지만, 인수 테스트를 자동화하기 위한 토대가 될 수 있습니다. BDD로 작성된 시나리오(Gherkin과 같은 구문으로)는 일반적으로 테스트 자동화 프레임워크와 연결됩니다. 이러한 접근 방식 중 하나 이상을 사용하여, 예를 들어 Gherkin 시나리오를 상세화하기 위한 자동화된 테스트 스크립트가 테스트 구현 활동의 일부로 생성될 것입니다. 테스트 스크립트는 충분한 이해를 위한 주석을 포함하고 좋은 코딩 프랙티스를 사용함으로써 테스트 문서 역할도 수행합니다. 일반적으로 개발된 테스트 스크립트를 그룹화하기 위해 테스트 스위트가 생성됩니다. 이 스위트는 이후 CI/CD 파이프라인의 테스트 작업과 연결됩니다.

테스트 분석 및 설계 활동 중에 명시된 특정 테스트 데이터는 가급적 재배포 가능한 형식으로 생성됩니다. 테스트 데이터는 때때로 코드로서 설정되며, 여기서 테스트 데이터는 스키마 파일 및 규칙 형식으로 정의됩니다.

CI/CD 파이프라인 내에서 구현되도록 인수 테스트가 명세됩니다. 때때로 신뢰 테스트(confidence test) 또는 스모크 테스트(smoke test)라고 불리는 이 테스트는 파이프라인의 모든 테스트 실행 작업 시작 시 테스트 객체가 상세하고 추가적인 테스트를 위한 준비가 되었는지 여부를 결정하는 데 사용됩니다. 스모크 테스트는 테스트 실패 시 개발 팀과 이해관계자에게 즉각적인 피드백을 제공하기 위함입니다. 또한, 자동화된 배포 상태 점검표(deployment health checklist)가 CI/CD 파이프라인 내의 모든 테스트 환경/단계에 대해 정의되는 경우가 많습니다.

테스트 데이터 준비, 테스트 환경 변경 또는 테스트 자동화 구현과 관련된 작업은 반복 계획 동안 팀의 작업 보드에 추정되고 계획됩니다. 특정 프랙티스 2.4 테스트 실행 일정 개발과 관련하여, 자동화된 테스트 실행은 CI/CD 파이프라인 내의 모든 코드 변경에 의해 트리거됩니다. 변경 시 테스트 실행을 시작하도록 트리거 시나리오가 식별되고 코딩됩니다 (웹 후). 수동 테스트 실행은 반복 계획과 연계하여 일정화되며, 작업 보드를 통해 테스트 작업으로 관리되고 일일 스탠드업 회의 동안 논의됩니다.

2.4.3 SG3 테스트 실행 수행 (Perform Test Execution)

테스트는 모든 변경 사항에 대해 명세된 CI/CD 파이프라인 작업에 따라, 그리고 반복 계획을 따르는 팀이 개발하는 새로운 스토리에 대해 실행됩니다. 테스트 작업의 순서는 빠른 피드백을 제공하도록 정의됩니다. 충분한 리소스가 있다면 피드백을 증폭시키기 위해 테스트 작업은 병렬로 실행될 수 있습니다. 클라우드 비용과 지속 가능성은 CI/CD 파이프라인을 최적화하는 데 핵심입니다.

결함/인시던트가 보고되고 테스트 로그가 생성됩니다. DevOps 프로젝트에서 소프트웨어는 코드 변경에 의해 트리거되어 가능한 한 자주 프로덕션에 전달됩니다. 테스트는 품질 좋은 제품을 전달하기 위해 협력하여 작업이 수행되는 프로세스의 필수적인 부분입니다.

스모크 테스트는 더 큰 테스트 세트가 실행되기 전에 CI/CD 파이프라인에 정의된 테스트 작업의 일부로 실행됩니다. 실패할 경우, 파이프라인의 실행은 다음 작업으로 계속되지 않으며 개발팀에 피드백이 제공됩니다. 실패 원인이 결함으로 식별되면, 결함이 보고됩니다.

테스트 실행은 반복 계획 동안 정의된 우선순위에 따라 수행됩니다. 일부 테스트는 문서화된 테스트 절차를 사용하여 실행될 수 있지만, 일반적으로 많은 테스트는 탐색적 및 세션 기반 테스트를 프레임워크로 사용하여 실행될 것입니다. 탐색적 테스트 세션은 미리 정의된 테스트 헌장(test charters)을 기반으로 하며, 시간 제한이 있습니다.

테스트는 소프트웨어가 모든 사용자에게 릴리스되기 전에 프로덕션에서도 실행될 수 있습니다. 크라우드 테스트(crowd testing)의 경우, 미리 선택된 베타 테스터 그룹이 대규모 릴리스 전에 제품의 특정 품질 측면에 대한 피드백을 제공합니다. 이를 위해서는 기능이 사용자 또는 지리적 위치와 같은 사전 정의된 전제 조건에 따라 켜거나 끌 수 있도록(feature toggle) 요구됩니다.

DevOps 개발에서는 회귀 테스트를 조직하고 구조화할 필요성이 증가합니다. 특히 단위 및 통합 테스트 수준에서의 회귀 테스트는 지속적인 통합 프로세스의 일부입니다. 이는 기본적으로 모든 코드 변경 시 발생하는 자동화된 "빌드 및 테스트" 프로세스이며, 결함을 조기에 신속하게 감지합니다. 지속적인 통합은 자동화된 회귀 테스트를 정기적으로 실행하고 코드의 품질 및 달성된 코드 커버리지에 대한 빠른 피드백을 팀에 제공합니다. 테스트는 또한 코드 변경에 의해 트리거되는 명세된 CI/CD 파이프라인 작업에 따라 실행됩니다. 이러한 자동화된 테스트 스크립트는 테스트 결과와 테스트 로그를 제공할 것입니다. 집계된 테스트 로그는 일반적으로

에픽, 사용자 스토리, 테스트 케이스, 테스트 스위트, 테스트 작업 및 전체 파이프라인에 대한 보기를 제공할 것입니다.

실제 결과와 예상 결과 간의 불일치는 결함으로 보고됩니다. 결함 관리는 일반적으로 개발 시스템의 필수적인 부분이며 실행 및 코드 변경에 대한 완전한 추적성을 제공합니다. 또한, 테스트 로그는 테스트 실행에 대한 관련 세부 정보의 연대기적 기록을 제공하기 위해 모든 시스템을 통해 자동으로 제공됩니다.

팀에 의해 새로운 사용자 스토리가 개발되고 테스트됨에 따라, 발견된 모든 결함을 실제로 로깅해야 하는지에 대해 애자일 프로젝트 내에서 일반적으로 논의가 있습니다. 특히 팀이 함께 위치(co-located)하는 경우, 테스터는 개발자와 대화해야 하며, 즉시 수정되어 다음 빌드에 포함될 수 있는 결함은 로깅할 필요 없이 수정되기만 하면 될 수 있습니다. 일부 팀은 반복을 벗어나는 결함만 로깅하고, 일부는 당일 수정할 수 없는 경우에만 로깅하며, 일부는 높은 우선순위 결함만 로깅합니다. 발견된 모든 결함이 로깅되지 않는다면, 어떤 결함을 로깅해야 하고 어떤 결함을 로깅해서는 안 되는지를 결정하는 기준이 있어야 합니다.

테스트를 거친 항목이 정의된 인수 기준을 충족하고 실제로 "완료(done)"로 분류될 수 있는지 판단하기 위해 테스트 실행 중에 데이터를 로깅하는 것은 DevOps 팀에서도 좋은 프랙티스로 간주됩니다. 테스트 데이터 로깅은 사용 중인 자동화에 의해 지원되어야 합니다. 이는 탐색적 테스트와 같은 경험 기반 기법을 적용할 때도 마찬가지로 사실입니다.

문서화하는 것이 유용할 수 있는 정보의 예는 다음과 같습니다:

- 테스트 커버리지 (얼마나 커버되었고, 얼마나 남았는지)
- 테스트 중 관찰 사항 (예: 테스트 중인 시스템과 사용자 스토리가 안정적으로 보이는지)
- 위험 목록 (어떤 위험이 커버되었고 어떤 위험이 가장 중요하게 남아 있는지)
- 발견된 결함 및 기타 문제
- 가능한 미해결 질문

로깅된 정보는 팀과 이해관계자가 수행된 모든 테스트의 현재 상태를 쉽게 이해할 수 있는 방식으로 일부 형태의 상태 관리 도구 (예: 테스트 관리 도구, 작업 관리 도구, 작업 보드)에 기록 및/또는 요약되어야 합니다. 물론 팀이 상태를 알아야 하는 것은 격리된 테스트 작업의 상태만이 아니라, 사용자 스토리의 전반적인 상태입니다. 백로그 항목은 완료 정의(Definition-of-Done)를 충족할 때 완료됩니다. DevOps 팀이 지속적인 배포(Continuous Deployment)를 프랙티스로 사용할 때, 완료 정의에는 일반적으로 성공적인 배포도 기준으로 포함됩니다.

2.4.4 SG4 테스트 인시던트 종결 관리 (Manage Test Incidents to Closure)

DevOps 에서 "인시던트(incident)"라는 용어는 일반적으로 사용자에게 영향을 미치는 실제 프로덕션 인시던트를 나타내는 데 사용됩니다. 따라서 이 문단의 나머지 부분에서는 TMMi 의 맥락에서 테스트 인시던트를 나타내기 위해 "결함(defect)"이라는 용어를 사용할 것입니다.

DevOps 팀은 운영(Operations)에 대한 책임도 있으므로 신속하게 대응해야 합니다. DevOps 내의 인시던트 관리는 IT 서비스 관리(ITIL) 프로세스입니다 : "정상적인 서비스 운영을 가능한 한 신속하게 복원함으로써 인시던트의 부정적인 영향을 최소화하는 프랙티스"입니다.

DevOps 에서는 거의 모든 것이 코드로 작성되기 때문에, 발견되는 모든 결함은 실제로는 어떠한 소프트웨어 내의 결함입니다. 환경 문제라면 그것은 환경을 코드로 구현한 인프라 코드(Infra as Code) 내의 결함이며,

잘못된 테스트 데이터라면 테스트 자동화 코드의 결함이고, 기타 실행의 불안정성이라면 파이프라인 코드의 결함일 수 있습니다.

이전 문단에서 언급했듯이, DevOps 환경에서는 발견된 모든 결함이 로깅되지는 않을 것입니다. 이 구체적인 목표는 로깅되어 종결될 때까지 관리되어야 하는 결함에만 적용됩니다.

원칙적으로 DevOps 에서 인시던트 관리는 간단합니다. 인시던트가 작업 보드에 로깅된 경우, 이는 스토리가 완료되는 것을 막는 결함으로 시각화됩니다. 표준 DevOps 작업 방식은 진행을 막는 다른 장애물이나 작업과 마찬가지로 적용될 것입니다. 이는 스탠드업 회의 중에 논의되고 팀에 의해 해결되도록 할당될 것입니다.

발견된 결함을 다른 반복으로 이월하기로 결정하는 경우, 이들은 단순히 제품에 대한 또 다른 원하는 옵션 (또는 변경 사항)이 됩니다. 이들은 백로그에 추가되고 그에 따라 우선순위가 지정됩니다. 우선순위가 충분히 높게 설정되면, 팀은 다음 반복에서 이를 처리할 것입니다.

발견 및 해결된 결함 수와 다음 반복의 백로그 일부가 될 가능성이 있는 미해결 결함 수를 일일 (스탠드업) 회의와 회고 회의 동안 모니터링하는 것은 좋은 프랙티스입니다.

2.5 프로세스 영역 2.5 테스트 환경 (Test Environment)

테스트 환경 프로세스 영역의 목적은 테스트를 관리 가능하고 반복 가능한 방식으로 실행할 수 있는, 테스트 데이터를 포함한 적절한 환경을 수립하고 유지하는 것입니다.

테스트 환경 프로세스 영역을 통해 테스트 데이터를 포함하여, 테스트를 관리 가능하고 반복 가능한 방식으로 실행할 수 있는 적절한 테스트 환경이 수립되고 유지됩니다. 물론 DevOps 소프트웨어 개발 프로젝트에서도 적절한 테스트 환경은 필수 불가결합니다.

모든 CI/CD 파이프라인 단계에서 클라우드 기술을 사용하면, 통제된 상태에서 요청 시 환경을 설정할 수 있습니다. 반복(iteration)의 주기가 짧기 때문에 테스트 환경은 매우 안정적이고 가용성이 높아야 합니다. 테스트 환경, 예를 들어 CI/CD 파이프라인의 문제점은 항상 반복의 진행 및 결과에 즉각적인 영향을 미칠 것입니다. 따라서 테스트 환경 및 테스트 데이터의 구성과 변경을 적절하게 관리하는 것도 최대한 중요합니다.

2.5.1 SG1 테스트 환경 요구사항 개발 (Develop Test Environment Requirements)

테스트 환경 요구사항 명세는 프로젝트 초기에 수행되며, 이후 지속적인 학습을 기반으로 계속됩니다. 환경 필요성을 결정하고, 이후 환경 구현을 주도할 요구사항을 개발하기 위해서는 제품 가치 흐름, 프로덕션 환경, 기술 환경 및 사용 중인 개발 도구 체인을 이해하는 것이 중요합니다.

요구사항 명세는 그 정확성, 적합성, 실현 가능성 및 '실제 운영' 환경의 정확한 표현을 보장하기 위해 검토됩니다. 초기에 요구사항을 명세하는 것은 가상화, 컨테이너화 및 DevOps 환경에서 자주 사용되는 기타

클라우드 네이티브 측면을 기반으로 필요한 테스트 환경을 확보 및/또는 개발할 수 있는 더 많은 시간을 제공한다는 이점이 있습니다.

빌드 파이프라인(CI 부분)의 환경은 일반적으로 작고 단순한 규모로 배포되며, 증가하는 성능 및 신뢰성 기대치에 맞춰 지속적으로 개발됩니다. 그러나 릴리스 파이프라인(CD 부분)에서는 종단 간(end-to-end) 비즈니스 프로세스 테스트가 필요할 수 있으며, 그에 따라 훨씬 더 복잡한 테스트 환경이 필요할 수 있습니다. 테스트 환경 필요성은 인프라 코드화(Infrastructure-as-Code) 및 구성 코드화(Configuration-as-Code) 프랙티스 및 CI/CD 플레이북 또는 플랫폼 팀이 정의한 표준에 따라 백로그 정제(refinement) 세션 동안 식별되고 명세됩니다. 테스트 환경 및 데이터 요구사항의 우선순위는 해당 백로그 항목의 우선순위를 기반으로 합니다.

때로는 더 복잡한 환경 요구사항을 정의하기 위해 인수 기준을 포함하는 별도의 기술 스파이크(technical spike)가 명세되기도 합니다. 요구사항은 일반적으로 조직의 CI/CD 프랙티스 및 용어와 일치하는 인프라/구성 코드화 호환 언어로 작성됩니다. 요구사항은 가치 흐름 또는 CI/CD 파이프라인 단계에 매핑됩니다. 개발 도구 체인은 운영 및 배포 기술과 정렬되어 필요하고 완전한지 확인합니다.

2.5.2 SG2 테스트 환경 구현 수행 (Perform Test Environment Implementation)

배포 파이프라인과 프로덕션 및 테스트 환경 구현은 가능한 한 빨리 시작되어 첫 번째 반복(iteration)이 시작될 때 초기 버전의 환경을 사용할 수 있도록 해야 합니다. 테스트 환경은 주어진 CI/CD 테스트 단계에서 관련 테스트 레벨을 수행하기 위한 일반적인 테스트 데이터를 포함하는 프로덕션과 유사한 환경이어야 합니다.

효율적인 DevOps 팀은 프로덕션 자동화를 테스트 환경에도 재사용하여 요청 시(on demand) 배포합니다. 환경 구현과 관련된 모든 작업은 다른 백로그 항목과 마찬가지로 백로그 내에서 처리되어야 합니다.

DevOps 환경에서는 종종 인프라 코드화(Infrastructure-as-Code)가 환경 구성 요소를 정의하고 구현하는 데 사용되며, 구성 코드화(Configuration-as-Code)는 환경 변수와 구성을 정의하고 구현하는 데 사용되는데, 이 둘 모두 자동화된 스크립트와 파이프라인을 통해 이루어집니다. CI/CD 파이프라인 내에서 셀프 서비스를 사용하여, 특히 탐색적 테스트 세션을 위해서도, 요청 시 환경을 프로비저닝(on-demand environment provisioning)하는 기능이 중요한 경우가 많습니다.

코딩 환경을 위한 일반적인 테스트 데이터는 자동화된 테스트 데이터 생성 및 관리 도구를 사용하여 생성됩니다. 스모크 테스트는 일반적으로 환경 인스턴스를 배포한 후, 그리고 해당 환경이 추가 활동에 사용되기 전에 파이프라인 내에서 실행됩니다.

2.5.3 SG3 테스트 환경 관리 및 통제 (Manage and Control Test Environments)

테스트 및 프로덕션 환경의 관리 및 통제는 DevOps 프로젝트 전반에 걸쳐 이루어질 것입니다. 환경은 테스트를 중단 없이 실행할 수 있도록 관리 및 통제되며, 사용 가능한 도구를 활용하여 환경과 데이터를 생성, 구성, 할당 및 사용할 수 있도록 합니다.

프로비저닝, 구성, 할당, 사용 및 서비스 해제를 지원하기 위해 테스트 환경에서 수행되는 시스템 관리는 다음 프랙티스들로 구성됩니다:

- 코드, 자동화 및 모니터링을 통한 인프라 관리.

- 로그인 세부 정보를 제공하여 테스트 환경에 대한 접근 관리.
- 테스트 실행 중 진행을 방해하는 문제에 대한 기술 지원 제공.
- 테스트 결과 및 결함 원인 분석에 사용될 수 있는 모니터링 및 텔레메트리 제공.

또한, 테스트 프로세스를 지원하기 위해 테스트 데이터의 프로비저닝, 할당, 수정 및 사용이 관리되며, 가능할 경우 자동화됩니다.

테스트 환경의 생성 및 할당은 최대의 가용성과 효율성을 달성하기 위해 자동화, 조정 및 측정됩니다. 특히 영구적인 환경에 대해서는 셀프 서비스 또는 요청 시 환경 프로비저닝(spin-up/down) 및 환경 인스턴스 모니터링이 구현됩니다.

테스트 환경을 프로비저닝하거나 사용할 때 발생하는 문제들은 DevOps 팀에 의해 로깅되고 종결될 때까지 관리됩니다. 테스트 환경 인스턴트는 문제가 관찰될 때, 자동으로 실패한 테스트의 일부로 또는 환경 사용자에게 의해 로깅됩니다. 테스트 환경 프로세스 영역에 대한 주요 결론은, 구체적인 목표와 프랙티스가 모두 여전히 적용 가능하며 본질적으로 변하지 않는다는 것입니다.

3 TMMi Level 3 정의(Defined)

3.1 프로세스 영역 3.1 테스트 조직(Test Organization)

테스트 조직 프로세스 영역의 목적은 테스트를 담당하는 고도로 숙련된 인력 그룹을 식별하고 조직하는 것입니다. 테스트 외에도, 이 테스트 그룹은 조직의 현재 테스트 프로세스 및 테스트 프로세스 자산의 강점과 약점에 대한 철저한 이해를 바탕으로 조직의 테스트 프로세스 및 테스트 프로세스 자산에 대한 개선을 관리하는 역할도 수행합니다.

3.1.1 SG1 테스트 조직 수립(Establish a Test Organization)

테스트 조직은 종종 오해되는 프로세스 영역입니다. 많은 사람들은 TMMi가 독립적인 테스트를 수행하는 독립된 테스트 그룹 또는 부서를 요구한다고 해석합니다. 그러나 DevOps 팀은 흐름 최적화와 피드백 증폭에 중점을 둔 교차 기능적(cross-functional)으로 설계되었기 때문에, 독립적인 테스트 조직은 이 개념을 저해할 것입니다. 따라서 본 문서에서 언급하는 고숙련 테스트 전문가들은 일반적으로 여러 교차 기능 팀(Cross-functional team)에 분산되어 다양한 역할을 수행합니다. 일반적으로 테스트 역량 센터(test competence centre) 또는 테스트 길드(test guild)는 기술 및 경력 경로를 포함하여 조직의 테스트 프로세스 및 전문가 직업을 개선하는 데 중점을 두는 반면, 장기간 지속되는 스크럼 팀은 가치 전달 최적화에 중점을 둡니다. 이 두 가지 모두 개선을 위한 메커니즘이 됩니다.

소위 테스트 역량 센터(test competence centre)의 전형적인 임무와 책임은 다음과 같습니다:

- 조직 전체의 표준 테스트 프로세스 수립, 관리 및 개선.
- 테스트 경험, 지식 및 기술을 갖춘 구성원을 교차 기능적 DevOps 팀에 위임.

테스트 역량 센터는 일반적으로 테스트 방법론 및 프로세스에 대한 소유권을 가집니다. 테스트는 일상적으로는 일반적으로 DevOps 팀의 일부이며, 따라서 CI/CD 파이프라인의 지속적인 테스트에 참여합니다. 하지만 그 외에도 테스트가 테스트 전문가 직업에 중점을 둔 테스트 역량 센터나 테스트 길드와 같은 테스트 조직에 속하는 것이 일반적입니다.

테스트 조직은 테스트 및 품질 문제와 관련된 영역에서 리더십을 요구합니다. 하지만 테스트는 가치 흐름(value stream)의 필수적인 부분으로 간주되어야 하며, 테스트 관련 활동은 DevOps 조직의 다양한 역할에 분산되어 있습니다. 필요한 역할에 테스트 기술을 확립하기 위해, 테스트 전문가에 의한 품질 지원(quality assistance)이 적용됩니다. 테스트 역량 센터(test competence centre) 옵션은 DevOps와 잘 맞으며, 테스트가 하나의 전문 분야(discipline)로 보존되고 진지하게 다루어지도록 보장합니다. DevOps 환경에서 테스트 조직이 취할 수 있는 또 다른 옵션은 테스트 길드(test guild) 형식입니다. 테스트 길드는 일반적으로 다양한 DevOps 팀의 구성원인 테스터들로 이루어진 비공식적인 자율 관리 그룹입니다. 테스트 길드의 중요한 활동으로는 지식 공유, 그룹별 학습, 트렌드 관찰 등이 있을 수 있습니다. 길드 회원은 자발적인 기준으로 가입할 수 있습니다. 테스트 길드에는 테스트 경력 경로 수립, 교육 조직, 테스트 프로세스 개선 결정/계획/구현과 같은 더 공식적인 임무가 할당될 수도 있습니다. 테스트 길드가 성공하기 위한 중요한 제약 조건은 구성원들에게 수행할 다양한 길드 활동에 시간이 할당되어야 한다는 것입니다.

3.1.2 SG2 테스트 전문가를 위한 기능 수립(Establish Test Functions for Test Specialists)

테스팅은 가치 있는 직업이자 전문 분야로 간주됩니다. DevOps 팀에게는 테스트 지식과 기술이 필요합니다. 테스트 전문가는 이러한 지식과 기술을 제공하며, 테스트 활동을 수행하는 동안 다른 팀 구성원들을 코치할 것입니다. 전형적인 DevOps 테스터에게 요구되는 지식과 기술은 전통적인 조직과 다릅니다. 테스터는 당연히 여전히 "전통적인" 테스트 지식과 기술이 필요하지만, 추가적으로 일반적으로 테스트 자동화에 대한 지식과 기술이 필요하며, 예를 들어 스크립팅이나 요구 사항 엔지니어링과 같이 테스트 외의 작업도 수행할 수 있어야 합니다. DevOps에서는 행동 주도 개발(BDD)과 인수 테스트 주도 개발(ATDD)이 인기 있는 접근 방식이므로, 이러한 기술 역시 DevOps 테스터의 기술 세트에 포함되어야 합니다. 더불어, CI/CD 파이프라인 설계 및 구성, 배포 자동화, 구성 관리 및 배포 전략 또한 테스터 역량의 일부가 되어야 합니다. DevOps 테스터는 팀 내에서 일하기 때문에, 소프트 스킬 역시 기술적 기술을 증진시키는 것만큼이나 중요합니다. DevOps 테스터는 소위 T 자형 인재(T-shape tester)로, 여러 개의 강력한 지식 영역을 가지고 있으며, 이는 테스트 기능 설명에 반영되어야 합니다. 일부 테스터는 기술에 더 강할 수 있는 반면, 다른 테스터는 설계 사고(design thinking)와 고객 이해에 더 집중할 수 있습니다. 일부 DevOps 기반 조직은 세 가지 유형의 테스트 기능을 생성했습니다:

- SDET (Software Development Engineer in Test): 테스트 및 배포 자동화 솔루션과 DevTest 구성원을 지원하는 제품의 테스트 용이성(testability) 기능을 개발합니다.
- DevTest: 제품 테스트 책임도 지는 개발자로서, 요구 사항을 분석하고, 테스트를 설계하며, 구현 및 실행할 수 있습니다 (일반적으로 SDET 가 제공하는 가용한 테스트 자동화 도구 및 테스트 용이성 기능을 활용합니다).
- 테스터 (또는 테스트 아키텍트, Test Architect): DevOps 팀 간의 테스트 활동을 감독하고, 위험을 식별 및 관리하며, 필요한 경우 제품에 초점을 맞춘 전문화된 테스트를 실행하기도 하지만, 주로 전반적인 테스트 노력에 대해 정보를 제공받는 역할을 합니다.

테스트 조직은 어떤 형태를 취하든, DevOps 환경에서 유능한 테스터가 될 기술과 동기를 갖춘 사람들로 인력이 배치됩니다. 이들은 특정 테스트 기능(위의 예시 참조)에 할당됩니다. 특히 DevOps 환경에서는 팀 전체가 테스트 활동을 수행할 것이기 때문에, 다른 비테스트 관련 기능 설명에도 그들의 테스트 활동, 역할, 책임, 그리고 요구되는 테스트 지식과 기술에 대한 기대치를 상세히 기술하는 것이 매우 중요합니다.

3.1.3 SG3 테스트 경력 경로 수립(Establish Test Career Paths)

테스트 경력 경로는 테스트 전문가들이 그들의 지식, 기술, 지위, 그리고 보상을 개선하여 경력을 발전시킬 수 있도록 확립됩니다. 예를 들어, DevOps 테스트 엔지니어에서 DevOps 테스트 아키텍트, 테스트 코치로 성장하거나, 고객 이해 기술을 바탕으로 제품 책임자(product owner)가 되거나, 대인 관계 기술을 바탕으로 스크럼 마스터(Scrum Master)가 될 수 있습니다. 이 경력 경로는 테스트 영역에 국한될 필요가 없으며, 이는 장기적으로 조직 전체에 품질 마인드셋을 확산하는 데 도움이 될 것입니다.

정의된 테스트 경력 경로를 기반으로, 테스트 조직의 모든 구성원을 위해 개인별 테스트 경력 개발 계획이 개발되고 유지됩니다. 이 구체적인 목표와 그에 수반되는 특정 프랙티스는 DevOps 환경에서도 완전히 적용

가능합니다. DevOps 팀은 그들의 도전적인 업무를 이해하고, 다른 팀 구성원을 코치할 수 있으며, 동기 부여된 테스터를 필요로 합니다. 이러한 요구 사항의 상당 부분은 테스트 경력 경로를 통해 다루어집니다.

하지만 테스트 경력 경로의 실제 정의는 순차적 라이프사이클을 따르는 조직과 비교할 때 DevOps 조직에서는 일반적으로 다소 다를 것입니다. 전통적인 조직에서는 테스트 경력 경로가 종종 수직적 성장 (테스터에서 테스트 관리자)에 초점을 맞추는 반면, DevOps 조직의 테스트 경력 경로는 수평적 성장 (주니어 테스터에서 시니어 테스터, 테스트 아키텍트, 그리고 아마도 테스트 코치로) 또는 스크럼 마스터나 제품 책임자와 같은 역할로의 성장에 초점을 맞추는 경향이 있습니다.

3.1.4 SG4 테스트 프로세스 개선 결정, 계획 및 구현(Determine, Plan and Implement Test Process Improvements)

테스트 개선은 DevOps 조직의 지속적인 개선 사고방식 과 린(lean) 원칙 및 시스템 사고에 초점을 맞춘 가치 흐름 관리(Value Stream Management) 프레임워크의 근본적인 부분입니다.

- 주요 아이디어는 다음을 중심으로 그룹화됩니다:
- 흐름 증폭(amplifying flow)
- 낭비 제거(removing waste)
- 피드백 증폭(amplifying feedback)
- DevOps 파이프라인에서 수집된 사실 기반 구축, 예를 들어 흐름 매트릭스 또는 네 가지 DORA 성능 매트릭스

다른 개선 사항들과 마찬가지로, 테스트 개선 사항도 조직의 백로그에 캡처되고, 우선순위가 지정되며, 관리됩니다.

이 특정 TMMi 목표와 그 특정 프랙티스가 설명하는 개선 프로세스는 주로 DevOps 팀이 수행하는 회고 회의를 통해 다루어지며, 파이프라인에서 생성된 데이터를 활용하는 사실과 데이터에 의존합니다. 회고는 DevOps 팀이 스스로를 점검하고 개선을 위한 조치를 정의할 수 있는 기회입니다. 회고는 일반적으로 반복 검토 후 그리고 다음 반복 계획 전에 발생합니다. 파이프라인의 일부로서, 설계 메트릭이 식별되고 데이터 수집이 확립되어, 지연, 제약 또는 제품 품질 문제가 발생하는 지점을 이해하기 위해 가치 흐름 맵에 시각화될 수 있습니다. 이러한 메트릭은 (테스트) 개선을 위한 기회를 식별할 수 있게 합니다. DevOps 팀은 테스트 실행 시간, 회귀 테스트 실패율, 테스트 환경 배포 및 구성 시간과 같이 자신들과 더 가까운 활동을 측정하는 데 집중하는 반면, 더 큰 조직은 흐름 매트릭스와 네 가지 DORA 성능 매트릭스에 집중합니다.

회고(retrospective) 동안 팀은 (부분적으로 메트릭을 기반으로) 해당 반복에서 잘된 점, 개선될 수 있는 점, 그리고 다음 반복에서 개선하기로 약속할 내용에 대해 논의합니다. 이시카와(Ishikawa) 또는 5 Whys(5 가지 왜) 기법을 사용한 근본 원인 분석과 같은 문제 해결 기술도 문제를 분석하는 데 유용합니다. 지속적인 학습 문화는 개인이 DevOps 를 위한 지식을 지속적으로 습득하고 경험과 아이디어를 교환하도록 장려합니다. 각 회고 동안 DevOps 팀은 작업 프로세스 개선이나 팀 또는 가치 흐름 수준에서 준비 완료 정의(Definition of Ready) 또는 완료 정의(Definition of Done) 기준을 조정하는 것과 같이 제품 품질을 높이는 방법을 식별하고 계획합니다. 회고는 테스트 효과성, 테스트 생산성, 테스트 품질에 초점을 맞춘 테스트 관련 개선 결정으로 이어질 수 있습니다. 또한 애플리케이션, 사용자 스토리, 기능 또는 시스템 인터페이스의 테스트 용이성(testability)을 다룰

수도 있습니다. 테스터와 비테스터를 포함한 모든 팀 구성원은 테스트 및 비테스팅 활동 모두에 대해 의견을 제시할 수 있습니다. 회고가 끝날 무렵, DevOps 팀은 다음 반복에서 구현할 (테스트) 개선 사항을 식별해야 합니다. 다음 반복에서 이러한 개선 사항을 구현하는 것은 DevOps 팀 자체의 책임이므로, 개선 사항은 백로그 항목으로 관리됩니다. 테스터는 일반적으로 팀 내에서 개선 사항이 구현되도록 보장하기 위해 테스트 개선에 대한 소유권을 갖습니다. 회고는 팀이 프로젝트 수명 기간 동안 지속적으로 진화하고 개선할 수 있도록 하는 중요한 메커니즘입니다.

범위가 종종 반복(iteration)으로 제한되기 때문에, 주로 특정 팀 문제를 해결하는 데 중점을 둔 작지만 빈번한 개선이 이루어집니다. 이러한 개선의 초점은 종종 가치 흐름 간 학습(cross-value stream learning)이나 개선의 제도화에는 있지 않습니다. (테스트) 프로세스 개선이 조직되고 관리되는 방식을 볼 때, 조직 수준의 (테스트) 프로세스 그룹에는 덜 집중하고 팀의 자율 관리에 더 중점을 둘 가능성이 높습니다. 이것이 나쁜 것은 아니지만, 성공적인 로컬 테스트 개선 사항이 조직의 나머지 부분과 공유되고, DevOps 팀의 권한을 벗어나는 개선 영역도 다루어질 수 있는 메커니즘을 만드는 것이 중요합니다. 가능한 해결책은 테스트 조직 (테스트 역량 센터 또는 테스트 길드)의 대표가 로컬 회고 회의에 참석하여, 적절한 경우 개선 사항이 조직 전체에 공유되고 제도화되도록 보장하는 것일 수 있습니다.

대규모 조직에서는 종종 가치 흐름(value stream) 수준에서 전용 회고 이벤트를 실행하여 잠재적인 시스템 수준의 문제를 식별합니다. 테스트 프로세스 개선 담당자(test process improver)는 조직 전체의 테스트에 영향을 미치는 더 광범위하고 근본적인 문제나 이니셔티브를 파악하여, 이를 가치 흐름의 백로그에 추가할 수도 있습니다.

3.1.5 SG5 조직 테스트 프로세스 배포 및 교훈 통합(Deploy the Organizational Test Process and Incorporate Lessons Learned)

DevOps 프로젝트의 경우에도, 조직 표준 테스트 프로세스와 테스트 프로세스 자산을 필요하고 부가적인 가치를 제공하는 만큼 접근하고 재사용하는 것이 중요합니다. 이들은 당연히 모든 수준에서 백로그 항목의 품질 보증 및 테스트에 초점을 맞추어 DevOps 작업 방식과 정렬되어야 합니다. 테스트 조직은 이러한 것들이 모든 팀에 배포되도록 보장할 것입니다. (조직 표준 테스트 프로세스 및 테스트 프로세스 자산과 이것이 DevOps 프로젝트와 어떻게 관련되는지에 대한 더 자세한 정보는 테스트 라이프사이클 및 통합 프로세스 영역을 참조하십시오). 조직의 표준 테스트 프로세스 구현 및 DevOps 프로젝트에서의 테스트 프로세스 자산 사용은 자율 조직 팀(self-organizing teams) 스스로 모니터링할 수 있습니다. 예를 들어, 회고(retrospectives)에서 이 문제를 다룸으로써 가능합니다. 앞으로 나아갈 훌륭한 방법은 프로세스와 템플릿을 도구에 내장하고, 파이프라인에 의해 프랙티스가 의무화되도록 만드는 것입니다. 예를 들어, 커버리지 목표가 충족되지 않거나 합의된 코딩 표준이 위반되면 풀 리퀘스트를 자동으로 거부하는 것입니다.

테스트 조직의 대표자, 예를 들어 테스트 프로세스 개선 담당자가 이러한 회의에 참석하면, 이 사람은 구현 상태를 추적하고 프로젝트 간 문제를 다룰 수 있습니다. 또한 이 사람은 DevOps 팀으로부터 귀중한 교훈(lessons learned)과 테스트 개선 사항을 다시 테스트 조직으로 가져갈 수 있습니다. 이 교훈과 테스트 개선 사항은 이후 테스트 프로세스 개선 제안으로 제출되고 그에 따라 관리될 수 있습니다.

3.2 프로세스 영역 3.2 테스트 교육 프로그램(Test Training Program)

테스트 교육 프로그램 프로세스 영역의 목적은 사람들이 테스트 작업과 역할을 효과적이고 효율적으로 수행할 수 있도록 지식과 기술의 개발을 촉진하는 교육 프로그램을 개발하는 것입니다.

테스트 교육 프로그램의 주요 목표는 테스터뿐만 아니라 팀 내부 및 외부에서 테스트에 관여하는 다른 모든 사람들에게 필요한 (테스트) 교육을 제공하는 것입니다. 효과적인 교육 프로그램은 테스트에 관련된 사람들이 그들의 테스트 지식과 기술을 계속해서 향상시키고, 필요한 도메인 지식과 테스트 관련 기타 지식 및 기술을 습득하도록 보장합니다. 품질과 테스트는 팀의 책임이므로, 테스트 관련 교육이 테스트 전문가에게만 제공되는 것이 아니라 전체 DevOps 팀에게 제공될 것으로 예상됩니다.

3.2.1 SG1 조직 테스트 교육 역량 수립(Establish an Organizational Test Training Capability)

이 프로세스는 조직의 전략적인 테스트 교육 요구 사항을 식별하는 것으로 시작합니다. 전략적인 테스트 교육 요구 사항은 지속적인 통합 및 지속적인 전달 파이프라인 내에서 테스트 프랙티스를 효과적으로 구현하는 데 필요한 기술과 지식을 팀에 갖추게 하는 것 및 보안 고려 사항이 다루어지도록 보장하는 것을 중심으로 전개됩니다. 테스터를 위한 전통적이고 애자일 교육 요구 사항(예: 테스트 설계 기법, 탐색적 테스트 및 커버리지 측정에 관한 교육) 외에도, 이제 DevOps 테스트의 특수성에 대한 지식과 기술을 습득할 필요가 있습니다. 그 예시로는 테스트 자동화 도구 및 프레임워크, 보안 테스트 (예: 정적 애플리케이션 보안 테스트(SAST), 동적 보안 분석 테스트(DAST) 및 침투 테스트), 인프라 코드화(Infrastructure-as-Code), 쉬프트 레프트 테스트 프랙티스, 테스트 주도 개발(TDD), 행동 주도 개발(BDD), 그리고 인수 테스트 주도 개발(ATDD)이 포함됩니다. 전략적인 테스트 교육 요구 사항의 식별은 단순히 테스터에게만 국한되지 않습니다. DevOps 팀의 다른 구성원들에 대한 테스트 관련 교육 요구 사항을 명세하는 것 또한 가장 중요합니다.

다른 조직에서와 마찬가지로, 테스트 교육 요구 사항은 (간결한) 교육 계획으로 전환될 것이며, 이는 특정 프로젝트의 테스트 교육 필요 사항도 다를 수 있습니다. 이 교육 계획은 검토되어야 하며 약속(commitments)이 수립되어야 합니다. 특히 DevOps에서는 일부 기술이 비공식적인 수단 (예: 직무 교육(training-on-the-job), 점심 시간 교육 세션, 코칭 및 멘토링)을 통해 효과적이고 효율적으로 전달되는 반면, 다른 기술은 공식 교육을 필요로 한다는 점에 유의해야 합니다. 또한, 지식 내재화(Embed Knowledge) 및 지속적인 실험(Continually Experiment)과 같은 DevOps 프랙티스가 교육 프로그램 개발 및 전달에 사용될 접근 방식을 주도할 것입니다.

3.2.2 SG2 필요한 테스트 교육 제공(Provide Necessary Test Training)

일반적으로 SG2 "필요한 테스트 교육 제공"에 대한 구체적인 프랙티스는 전통적인 조직과 DevOps 조직에서 동일합니다. 조직 교육 계획을 기반으로, 자신의 테스트 역할을 효과적으로 수행하기 위해 교육을 받아야 하는 팀 구성원이 식별됩니다. 이후 필요한 리소스를 포함하여 교육 일정이 잡히고 실시됩니다.

위에 언급된 바와 같이, DevOps 조직에서는 비공식적인 교육 수단이 자주 사용됩니다. 예를 들어, 공식 교육 외에도, 현장 멘토링 및 코칭(on-the-job mentoring and coaching)이 일반적으로 지속적으로 장려되고

활용됩니다. 실제로, 멘토링은 모든 사람의 책임이며 조직의 모든 수준에서 기대됩니다. 짝을 이루어 작업하는 것(Working in pairs)은 기술 향상 및 지식 이전을 위해 팀이 흔히 사용하는 또 다른 수단입니다.

비공식적이든 공식적이든, 실시된 테스트 교육에 대한 기록이 생성될 것이며, 테스트 교육의 효과성이 평가됩니다. (테스트) 교육 참석은 회고 회의의 일부로도 평가될 수 있습니다. 이 회의에서는 참석자들의 지식과 기술이 그들의 테스트 작업을 수행하는 데 이제 더 적절한지 여부에 대한 논의가 이루어집니다.

3.3 프로세스 영역 3.3 테스트 라이프사이클 및 통합(Test Lifecycle and Integration)

테스트 라이프사이클 및 통합의 목적은 사용 가능한 조직 테스트 프로세스 자산 세트와 작업 환경 표준을 수립하고 유지하며, 테스트 라이프사이클을 개발 라이프사이클과 통합 및 동기화하는 것입니다. 통합된 라이프사이클은 전체 가치 흐름(value stream) 전반에 걸쳐 테스트의 조기 참여를 보장합니다. 또한 테스트 라이프사이클 및 통합의 목적은 식별된 위험과 정의된 테스트 전략을 기반으로 다중 테스트 레벨 (DevOps 환경에서는 CI/CD 파이프라인의 단계로 표현됨)에 걸친 일관된 테스트 접근 방식을 정의하고, 정의된 테스트 라이프사이클을 기반으로 전반적인 테스트 계획을 제공하는 것입니다.

3.3.1 SG1 조직 테스트 프로세스 자산 수립(Establish Organization Test Process Assets)

테스트 조직의 중요한 책임은 조직의 테스트 정책 및 목표에 부합하는 표준 테스트 프로세스를 정의, 문서화 및 유지보수하는 것입니다. 무엇보다도, 이 프로세스는 수행될 다양한 테스트 레벨 및 테스트 유형에 대한 설명을 포함합니다. TMMi는 상세한 문서화된 테스트 프로세스를 의무화하지 않으므로, 종종 예시로 상세화된 템플릿이 공통의 작업 방식을 수립하는 훌륭한 접근 방식이 됩니다. 팀이 프로젝트에서 적용하는 테스트 레벨과 테스트 유형에 대해 합의하는 애자일 테스트 사분면(Agile Testing Quadrants)(Agile Testing Quadrants)은 합의된 테스트 프로세스 및 전략을 나타내는 단순한 상위 수준 방식입니다. 의무적인 테스트 레벨은 이후 완료 정의(Definition of Done)에 기준으로 기록됩니다. CI/CD 파이프라인 내의 테스트 단계에 대한 상위 수준 보기도 좋은 공통 문서 역할을 합니다. 템플릿은 그 구조 내에 함축된 정보 수집 및 결과 문서화에 필요한 활동들을 식별합니다. 프로세스는 활동들의 엄격한 순서가 될 필요는 없습니다. 종속성이 존재하는 경우, 템플릿의 주석을 통해 기록될 수 있습니다. 템플릿은 프로세스의 진정한 의도를 전달하는 데 실질적인 가치를 가집니다. 또한 템플릿은 "장황한" 프로세스 문서에서 흔히 발견되는 모호성을 피할 수 있도록 합니다. DevOps 조직으로서 효과적인 프로세스를 가지고 있다면, 단순히 이를 가볍게 문서화하고 몇 가지 사항을 추가할 필요만 있을 수 있습니다. 일반적으로 DevOps 조직은 높은 수준의 자동화에 의존하며, 프로세스 및 규칙은 종종 도구에 캡처되고 의무화됩니다. 따라서 테스트 프로세스가 항상 별도의 문서로 존재하지 않을 수 있지만, 개발 및 배포 프로세스의 핵심적인 부분으로 남아 있습니다. 물론, 이러한 작업 방식에서도 관련된 모든 사람들이 적용되는 프로세스에 대해 공통된 이해를 갖는 것이 중요합니다. 테스트 프로세스 수립 시 DevOps 조직에서 자주 사용되는 몇 가지 주요 지침은 다음과 같습니다:

- 프로세스의 "필수 사항(must do's)"은 지침과 분리하여 패키징됩니다.
- 프로세스는 일반적으로 두 페이지를 넘지 않습니다.

- 프로세스는 모든 프로젝트의 규모와 관계없이 조직 전체에 이를 의무화하기로 결정하지 않는 한, "방법(how to)" 정보나 도구 정보를 포함하지 않습니다.
- 별도의 지침에는 테일러링(tailoring)/계획 옵션 및 "방법" 정보가 포함됩니다. "방법" 정보의 경우, 단순히 기존 서적이나 논문을 참조하도록 결정할 수도 있습니다.

대규모 조직은 플랫폼 엔지니어링 프랙티스를 사용하여 표준 백로그 항목 워크플로우 또는 CI/CD 파이프라인 워크플로우를 의무화할 수 있습니다. 이는 정의된 프로세스가 팀이 사용해야 하는 특정 도구에 구현됨을 의미합니다. 만약 테일러링(tailoring)이 가능하다면, 라이프사이클 단계는 선택 사항입니다. 많은 대규모 조직에서 정책, 프로세스/프랙티스, 작업 지침/절차, 그리고 이네이블러/템플릿과 같은 네 가지 수준의 프로세스 자산을 보는 것은 드문 일이 아닙니다. 이는 TMMi 가 요구하는 것이 아니라, 많은 대규모 조직이 프로세스 자산을 구현하기로 선택한 방식 때문입니다.

프로세스 자산의 선택은 각 조직에 달려 있지만, 많은 DevOps 조직은 이제 두 가지 수준의 프로세스 자산으로 충분함을 경험하고 있습니다. 이는 프로세스 수행 시 "무엇을 해야 하는지(what must be done)"를 요약하는

관련 프로세스 설명과 정책 선언문을 통합함으로써 달성됩니다. 두 번째 수준은 프로세스를 수행하고 맞춤화하는 "방법(how to)"에 대한 지침을 포함합니다. 이 수준은 프로세스를 맞춤화하기 위한 도움말로 간주될 수 있으며, 팀에게 정의된 프레임워크 내에서 정확히 어떻게 작업할지 결정할 수 있는 자율성을 부여하고, 일반적으로 지원 템플릿을 포함합니다. 대부분의 DevOps 조직에서 단계별 절차는 도구 가이드 및 교육/멘토링으로 대체됩니다. 위에서 언급했듯이, 테스트 현장 템플릿, 테스트 세션 시트 템플릿 또는 테스트 계획 템플릿과 같은 템플릿은 템플릿에 내포된 필요한 프로세스 활동을 가진 프로세스 역할을 할 수 있습니다. 이는 효과적인 애자일 프로세스 설명을 개발하기 위해 관찰되는 일반적인 기법입니다.

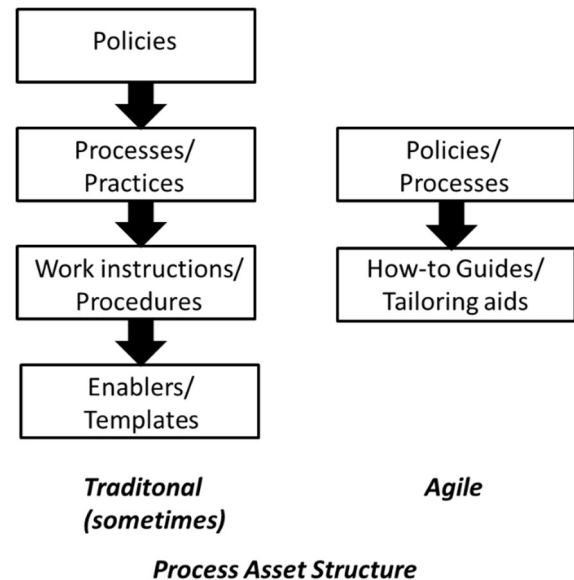


Figure 5: Levels of Process Assets

조직의 표준 테스트 프로세스 세트는 프로젝트나 팀에 의해 맞춤화(tailored)되어 그들의 구체적인 정의된 프로세스를 생성할 수 있습니다. SP 1.3 테일러링 기준 및 지침 수립의 맥락에서, 많은 조직이 소위 "테일러 다운(tailor down:전체에서 불필요한 것을 뺌)" 접근 방식을 사용합니다. 이 접근 방식은 사람들이 무엇인가가 필요하지 않은 이유를 설명하는 데 노력을 들이도록 요구합니다. 테일러 다운할 때, 어디까지 갈 수 있을까요? 최소한의 정의가 있을까요? 하지만, 최소한의 세트 (모두가 해야 하는 것)에서 시작한다면, "필수 사항"을 테일러링에서 제외할 위험을 제거할 수 있습니다. "테일러 업(tailoring up: 기본에서 필요한 것을 더함)" 접근 방식은 애자일 및 DevOps 접근 방식과 훨씬 더 일치하며, TMMi 를 완전히 준수합니다. 일련의 간단한 기준은 테일러 업할 때 올바른 질문을 하는 데 도움이 될 수 있습니다. 지원 기준이 없으면 일부는 너무 많이 테일러 업하는 경향이 있습니다.

표준 테스트 프로세스의 배포를 지원하기 위해, 테스터가 일상 업무에 도움을 받을 수 있는 템플릿, 모범 사례 및 체크리스트가 담긴 테스트 프로세스 자산 라이브러리가 생성되고 유지보수될 것이며, 이는 종종 위키에 구축됩니다. 일반적으로 이러한 템플릿은 CI/CD 파이프라인 청사진, 필수 필드 또는 백로그 항목 관리 도구의 템플릿과 같은 도구 구현의 일부입니다. 테스트 프로세스 자산 라이브러리를 생성하는 것은 팀이 테스트 활동

중에 제공된 자산을 사용하여 부가적인 가치를 얻는다면 DevOps 환경에서도 의미가 있습니다. 거의 동일한 논리가 테스트 프로세스 데이터베이스에도 적용되며, 여기에는 추정치, 커버리지, 품질 측정치와 같은 테스트 데이터가 수집되어 팀에 제공됩니다. 수집되고 제공되는 모든 것이 팀 간 학습 및 경험 공유에 도움이 되고 팀에 부가적인 가치를 제공하도록 보장하십시오. 물론 DevOps 환경에서도 작업 환경 표준은 프로젝트 작업 환경 생성을 안내하는 데 사용됩니다.

물론, DevOps 환경에서도 프로젝트 작업 환경 생성을 안내하기 위해 작업 환경 표준이 사용됩니다.

3.3.2 SG2 테스트 라이프사이클 모델을 개발 모델과 통합(Integrate the Test Lifecycle Models with the Development Models)

표준 테스트 라이프사이클 모델은 다양한 테스트 레벨에 대한 주요 활동 및 결과물을 정의합니다. 표준 테스트 라이프사이클 모델은 단계 설정, 마일스톤, 결과물 및 활동 측면에서 테스트 활동을 통합하기 위해 개발 라이프사이클 모델과 정렬됩니다. 라이프사이클 통합은 모든 백로그 항목에 대해 프로젝트에서 테스트의 조기 참여가 보장되는 방식으로 이루어집니다. 물론, DevOps에서는 개발과 테스트가 라이프사이클 내에 완전히 통합되어야 하며, 테스트는 가능한 한 빨리 이루어져야 합니다. 테스트가 팀의 일부이며 반복 계획에 참여하고, 위험이 사전에 식별되며, 테스트 활동에 합의함에 따라, 추정치와 약속이 영향을 받아 테스트가 라이프사이클에 통합되도록 보장합니다.

따라서 이 구체적인 목표는 DevOps 소프트웨어 개발을 수행하는 조직에도 적용 가능합니다. DevOps 환경에서는 개발과 테스트가 당연히 팀 수준에서 이미 완전히 통합되어 있어야 하며, 이것이 어떻게 수행되는지에 대한 공통의 이해가 있어야 합니다. 그러한 경우에는 추가적인 조치가 필요하지 않습니다. 다른 상황에서, 통합과 조기 참여가 아직 완전히 구현되지 않은 경우, 이 구체적인 목표는 매우 중요한 개선 영역이며, 반복(iteration)에서 수행되는 개발 및 테스트 활동을 훨씬 더 잘 정렬시키는 결과를 가져와야 합니다.

3.3.3 SG3 마스터 테스트 계획 수립(Establish a Master Test Plan)

TMMi 레벨 3에서는 테스트가 모든 테스트 레벨에 걸쳐 테스트 작업, 책임, 테스트 접근 방식의 조정을 다루는 마스터 테스트 계획 수립과 관련됩니다. 이는 다양한 테스트 레벨 간에 불필요한 중복이나 테스트 누락을 방지하며, 전체 테스트 프로세스의 효율성과 품질을 크게 향상시킬 수 있습니다. 마스터 테스트 계획은 수행될 특정 레벨과 그 레벨 간의 관계를 포함하여, 특정 프로젝트에 대한 테스트 전략의 적용을 설명합니다. TMMi 레벨 2에서는 테스트 계획 프로세스 영역의 일부로 릴리스 수준과 반복(iteration) 수준 모두에서 (테스트) 계획이 이미 다루어지고 있습니다. DevOps 팀은 일반적으로 여러 테스트 레벨을 수행하며, 이는 릴리스 및 반복 계획의 일부로서 모두 적절하게 다루어져야 합니다.

하지만 DevOps 라이프사이클 모델에 따라 작업하는 일부 프로젝트는 하이브리드 접근 방식을 채택했습니다. 즉, DevOps 팀 내의 테스트 외에도 일부 테스트가 팀 외부에서 조직되는 경우입니다. 이러한 상황에서 마스터 테스트 계획은 DevOps 팀과 그들의 범위를 벗어난 테스트 레벨 또는 유형(예: 아웃소싱된 비기능 테스트(보안, 성능) 또는 외부 당사자와의 시스템 통합 테스트) 간의 관계를 정의하고 관리할 것입니다. 이는 일반적으로 백로그 항목의 작업 보드에 추가적인 단계로 표현됩니다. 이 경우, 이 구체적인 목표와 관련된 모든 특정 프랙티스가 일반적으로 적용 가능할 것입니다.

모든 테스트가 DevOps 팀 내에서 이루어지는 경우, 특정 마스터 테스트 계획은 부가적인 가치가 없으므로, 이러한 경우에는 이 TMMi 목표가 관련성이 없을 가능성이 높습니다. 이 경우 릴리스 및 반복(iteration) 수준에서의 계획이 필요한 모든 테스트 측면을 커버할 것으로 예상됩니다. 그러나 팀이 지속적으로 합의하고 개선하는 일반적인 테스트 접근 방식을 가지고 있는지 확인하는 것은 유용합니다. 완료 정의(Definition of Done)에서 언급된 특정 테스트 레벨, CI/CD 파이프라인/워크플로우, 또는 단순한 애자일 테스트 사분면(Agile Testing Quadrants) 문서를 살펴보는 것은 이러한 것들이 간결한 마스터 테스트 계획으로 인식될 수도 있음을 의미합니다.

3.4 프로세스 영역 3.4 비기능 테스트(Non-Functional Testing)

비기능 테스트 프로세스 영역의 목적은 테스트 계획, 테스트 설계 및 실행 동안 비기능 테스트를 포함하도록 테스트 프로세스 역량을 개선하는 것입니다.

비기능 테스트가 중요한지 여부는 적용되는 라이프사이클과 무관합니다. 개발 중인 제품이 테스트에 관련되는 비기능적 측면을 결정할 것입니다. 이는 당연히 DevOps 개발에서도 마찬가지로 사실입니다. 따라서 비기능 테스트는 DevOps 소프트웨어 개발에도 적용 가능합니다.

하지만 비기능적 측면의 특성과 사용되는 테스트 접근 방식에 따라, 성능 및 신뢰성과 같은 일부 비기능적 특성은 전통적인 방식으로 짧은 반복(iteration) 내에서 테스트될 수 없습니다. 따라서 비기능적 속성을 테스트하기 위한 다른 접근 방식이 종종 필요합니다. 이것이 DevOps 에서 더 나은 방향으로 이루어진 근본적인 변화 중 하나인데, 품질 속성이 끝까지 미루어지지 않고 이미 초기에 테스트된다는 점입니다. 전체 비기능 테스트를 수행하기 위해 첫 번째 반복부터 모든 구성 요소/기능이 가용하지는 않겠지만, 이 테스트의 결과는 품질 문제에 대한 조기 징후를 제공하는 데 사용될 수 있습니다.

사용자 스토리는 사용자 및 고객을 위해 올바른 제품이 개발되도록 보장하기 위해 기능적 요소와 비기능적 요소를 모두 다루어야 합니다. ISO 25010 품질 특성은 요구 사항을 구조화하는 데 도움을 줄 수 있으며, 테스터는 이러한 비기능적 요소들을 고려해야 합니다. DevOps 에서는 표준 ISO 25010 품질 특성 외에도, 관찰 가능성(observability) (텔레메트리에 의해 지원됨)이 품질 특성으로 도입됩니다.

3.4.1 SG1 비기능 제품 위험 평가 수행(Perform a Non-Functional Product Risk Assessment)

테스트 계획 프로세스 영역의 일부인 특정 목표 SG1 위험 평가 수행에서 식별되고 설명된 DevOps 프로젝트를 위한 제품 위험 세션 및 기법(예: 리스크 포커, ROAM 및 위험 모델링)은 이제 비기능적 측면 및 위험을 명시적으로 포함하도록 확장될 것입니다. DevOps 에서는 종종 관찰 가능성(observability), 보안 및 규정 준수에 특별한 초점이 있습니다. 조직이 디지털화되고 클라우드 기반으로 전환됨에 따라 IT 시스템은 증가된 위험과 취약점에 직면합니다. 위험 모델링(Threat modelling)은 기능 라이프사이클의 초기 단계, 즉 분석/백로그 정제(refinement) 동안 수행되는 프로세스로, 위험 평가를 실시하여 위험을 식별하고 구조화된 모델로 완화하는 것을 목표로 합니다. 그 결과물은 종종 기능(feature) 수준의 비기능 인수 기준으로 반영되며, 추가 사용자 스토리 생성이나 전용 아키텍처 결정으로 이어질 수 있습니다. 반복(iteration) 수준에서는 사용자 스토리 수준에서 위험 평가가 수행되며, 이는 비기능 요구 사항을 주요 입력으로 정의하고 업데이트된 스토리 인수

기준으로 이어집니다. 제품 책임자를 포함하여 모든 팀 구성원과 가능하면 일부 다른 이해관계자들이 제품 위험 세션에 참여해야 합니다. 일부 비기능 영역에 대해서는 전문가의 도움이 필요할 수 있습니다.

제품 위험 세션은 일반적으로 우선순위가 지정된 비기능 제품 위험 목록을 문서화한 결과를 가져오며, 이는 기능 및 스토리 문서를 업데이트합니다. 이 제품 위험 목록은 CI/CD 파이프라인 및 비기능 테스트 접근 방식의 설계를 주도할 뿐만 아니라, 예를 들어 확장성 및 신뢰성을 위한 컨테이너화 사용과 같은 제품의 아키텍처 및 선택된 기술을 적극적으로 형성합니다. 테스트 계획 프로세스 영역에서 언급했듯이, DevOps 프로젝트에서는 사용되는 프로세스와 그 결과물인 문서 모두 순차적 라이프사이클 모델을 따르는 전통적인 프로젝트에 비해 훨씬 간소화된 형태를 가질 것입니다.

3.4.2 SG2 비기능 테스트 접근 방식 수립(Establish a Non-Functional Test Approach)

식별되고 우선순위가 지정된 비기능 제품 위험을 완화하기 위해 관련 비기능 품질 특성에 대한 테스트 접근 방식이 정의됩니다. CI/CD 파이프라인은 비기능 테스트도 다루도록 설계되어야 합니다. 특정 반복(iteration)에 대해서는 추가적인 테스트 대상 비기능적 측면이 반복 계획 중에 식별되는 반면, CI/CD 파이프라인의 비기능 테스트는 모든 변경에 대해 실행됩니다. 하지만, 비기능적 품질 특성은 관찰 가능성에 의존하는 텔레메트리를 사용하여 배포 후 프로덕션에서도 측정됩니다. 서비스 수준 지표(SLI)와 서비스 수준 목표(SLO)는 DevOps 에서 이를 관리하는 일반적인 방법입니다. 테스트할 비기능적 측면의 우선순위 목록은 일반적으로 해당 반복에서 개발 및 테스트되는 사용자 스토리와 관련됩니다. 반복 중에 새로운 비기능 제품 위험이 명백해질 수 있으며, 이는 추가적인 테스트를 요구합니다. 추가 테스트를 요구하는 새로운 비기능 제품 위험과 같은 문제들은 일반적으로 일일 스탠드업 회의에서 논의됩니다.

비기능적 위험을 완화하기 위해 반복(iteration) 수준에서 정의되는 비기능 테스트 접근 방식은 일반적으로 비기능적 위험의 수준과 유형을 기반으로 적절한 비기능 테스트 방법 및 테스트 기법(들)의 식별을 포함할 것입니다. 보통 이는 지원 도구의 사용과 비기능 테스트를 위한 테스트 자동화 접근 방식도 다를 것입니다. 릴리스 수준에서의 비기능 테스트 접근 방식은 훨씬 더 높은 수준일 것이며 정의된 CI/CD 파이프라인을 기반으로 해야 합니다. 릴리스 수준과 반복 수준에서의 비기능 테스트 접근 방식은 모두 전반적인 테스트 접근 방식의 일부이며 팀/프로젝트 위키에 보관되거나 표시될 것입니다. 비기능 테스트 결과는 제품 대시보드에 제시되어야 합니다.

비기능 종료 기준은 완료 정의(Definition of Done, DoD)의 일부입니다. DoD 는 예를 들어 API 응답 시간이나 "프론트엔드 웹 페이지가 OWASP Top 10 위험 목록에 대해 테스트되었다"와 같은 보안 기대치처럼 비기능 테스트와 관련된 특정 기준을 갖는 것이 중요합니다. CI/CD 파이프라인의 일부로서 배포 후에 측정되는 SLI 와 SLO 또한 종료 기준으로 사용되며, 일반적으로 SLI 또는 SLO 의 저하를 피하는 것이 목표입니다. 반복(iteration)은 합의된 비기능 인수 기준과 가능하면 사용자 스토리의 구현을 가져와야 하며, DoD 에 정의된 비기능 (테스트) 종료 기준을 충족해야 합니다. 반복 수준에서 비기능에 대한 DoD 가 존재할 뿐만 아니라, 종종 여러 반복에 걸쳐 있는 기능(feature) 또는 릴리스 수준의 DoD 도 있습니다. 또한, DevOps 환경에서는 배포 후 SLI 와 SLO 를 사용하여 지속적인 모니터링을 통해 프로덕션에서 비기능적 특성을 측정하는 Shift Right 접근 방식에 더 강하게 초점을 맞춥니다.

3.4.3 SG3 비기능 테스트 분석 및 설계 수행(Perform Non-Functional Test Analysis and Design)

이 구체적인 목표는 테스트 설계 및 실행 프로세스 영역의 특정 목표 SG1 테스트 설계 기법을 이용한 테스트 분석 및 설계 수행과 마찬가지로, 이제 비기능적 관점에서 주로 동일한 프랙티스를 따릅니다. 테스트 분석 및 설계 동안 비기능 테스트를 위한 테스트 접근 방식은 구체적인 테스트 조건과 테스트로 변환됩니다. DevOps 에서 테스트 분석 및 설계, 그리고 테스트 실행은 일반적으로 반복(iteration) 전반에 걸쳐 병렬로 실행되는 상호 지원적인 활동입니다. 이는 대부분의 비기능 테스트에도 해당됩니다. 따라서 비기능 테스트 분석은 명시적인 별도의 활동이 아니라, 테스트(DevOps 품질 엔지니어)가 협업적인 사용자 스토리 개발 및 다듬기 역할의 일부로서 수행하는 암묵적인 활동입니다.

비기능 인수 기준과 사용자 스토리 분석을 기반으로 비기능 테스트 조건이 식별됩니다. 테스트 조건은 기본적으로 테스트/커버해야 하는 "항목"의 식별입니다. 정의된 인수 기준이 충분히 상세하고 명확하다면, 일반적으로 전통적인 테스트 조건의 역할을 대신할 것입니다. 인수 기준은 이후 비기능 테스트로 변환됩니다. 비기능 테스트의 경우, 사용자 스토리만을 대상으로 하기보다는 더 높은 수준에서 테스트 분석을 수행하는 것이 종종 유익합니다. 예를 들어, 기능(feature)이나 에픽 또는 여러 스토리의 모음을 분석하여 사용자 스토리 수준보다 더 추상적이고 여러 사용자 스토리에 걸쳐 있는 비기능 테스트 조건을 식별하는 것입니다. DevOps 에서 테스트 우선(test-first) 원칙이 적용됨에 따라, 비기능 테스트 조건 세트를 커버하는 비기능 테스트는 코드 개발보다 이전에 또는 적어도 병렬로 식별되고 (그리고 가능하면 자동화됩니다).

대부분의 수동 비기능 테스트의 경우, 테스트는 팀이 비기능 테스트 실행을 진행함에 따라 식별/다듬어질 것입니다. 테스트는 전통적인 프로젝트에서처럼 상세하게 문서화되지 않는 경우가 많으며, 오히려 탐색적 테스트를 사용할 때 테스트 아이디어 형식으로 문서화됩니다. 복잡하고 중요한 비기능 영역의 경우, 공식적인 비기능 테스트 설계 기법을 사용하는 보다 전통적인 테스트 설계 및 테스트 케이스 개발 접근 방식이 위험을 커버하는 최선의 방법일 수 있습니다. 하지만 이 접근 방식에서도 전통적인 순차적 라이프사이클 환경에서의 비기능 테스트와 비교할 때 문서의 양은 제한될 것입니다.

관찰 가능성 및 텔레메트리 요구 사항 및 스토리를 식별하는 것도 비기능 분석 및 설계의 결과물이 될 수 있으며, 이는 프로덕션에서 비기능적 측면을 측정하는 데 도움이 됩니다. 비기능 테스트 실행을 지원하는 데 필요한 특정 테스트 데이터가 식별됩니다. DevOps 에서 필요한 테스트 데이터는 일반적으로 테스트 명세 문서에 먼저 완전히 명세되지 않지만, 예를 들어 페르소나(persona) 형태로 사용자 스토리의 일부로 기록되는 경우가 많습니다. 그러나 필요한 도구 및/또는 기능이 제공되는 경우, 비기능 테스트를 지원하는 데 필요한 테스트 데이터는 비기능 수동 테스트의 즉각적인 실행을 허용하기 위해 대부분 즉시 생성됩니다. 이와 대조적으로, 자동화된 비기능 테스트의 경우 데이터는 일반적으로 사전에 명세될 필요가 있습니다. 또한, 일부 비기능 요구 사항이 프로덕션에서 테스트되기 때문에, 테스트 데이터는 프로덕션 데이터일 수도 있습니다. 물론, 이러한 경우에는 개인 정보 보호 및 GDPR 이 핵심입니다.

비기능 요구 사항, 테스트 조건 및 테스트 간의 추적성은 수립되고 유지되어야 합니다. 팀은 테스트의 일부로서 다양한 비기능적 사용자 스토리와 인수 기준을 커버했음을 명확히 해야 합니다. 대부분의 DevOps 조직에서 사용자 스토리는 일련의 관련 인수 기준과 이후 테스트를 개발하기 위한 기초가 됩니다. 이 접근 방식을 사용하면, 요구 사항에서 테스트로의 수평적 추적성이 달성될 수 있습니다.

3.4.4 SG4 비기능 테스트 구현 수행(Perform Non-Functional Test Implementation)

테스트 구현은 테스트 실행을 시작하는 데 필요한 모든 것을 제자리에 배치하는 것에 관한 것입니다. 일반적으로 테스트 실행을 지원하기 위한 테스트 문서 개발, 예를 들어 테스트 절차 개발은 최소화됩니다. 대신, 자동화된 (회귀) 테스트 스크립트가 개발되고 우선순위가 지정됩니다. DevOps에서는 자동화할 수 있는 모든 것이 자동화되어야 합니다.

비기능 테스트 구현은 테스트 설계 및 실행 프로세스 영역의 특정 목표 SG2 테스트 구현 수행에서 이미 설명된 많은 프랙티스를 따를 것입니다. 구체적으로 무엇을 해야 하는지, 그리고 비기능 테스트 구현이 어떻게 이루어지는지는 정의된 접근 방식, 사용되는 기법, 그리고 어떤 비기능적 특성을 어느 수준까지 테스트해야 하는지에 따라 크게 달라집니다. 예를 들어, 준비가 덜 필요한 탐색적 테스트는 유용성 테스트에는 적합할 수 있지만, 광범위한 신뢰성 및 성능 테스트에는 덜 적합한 경우가 많습니다.

일부 비기능적 품질 특성의 경우, 테스트 데이터의 가용성이 필수적이며 테스트 구현 중에 생성될 필요가 있습니다. 이는 전통적인 프로젝트와 대체로 동일하지만, 문서화가 간결한 방식으로 이루어져 회귀 테스트를 위한 재사용이 여전히 가능하도록 한다는 점에서는 차이가 있습니다. 이전에 언급했듯이, 일부 비기능 요구 사항은 프로덕션에서 테스트되기 때문에, 테스트 데이터는 프로덕션 데이터일 수도 있습니다.

물론, 테스트 구현 및 준비는 다른 (테스팅) 활동과 병렬적으로 가능한 한 빨리 시작될 것입니다. 이것은 별도의 단계가 아니라, 효율적이고 효과적인 테스트 실행을 가능하게 하기 위해 수행되어야 하는 (작업 보드에 나열된) 일련의 활동입니다.

3.4.5 SG4 비기능 테스트 실행 수행(Perform Non-Functional Test Execution)

이 프로세스 영역의 이전 목표와 마찬가지로, 이 부분에서도 본 문서의 앞부분에서 논의된 테스트 설계 및 실행 프로세스 영역의 관련 특정 목표 SG 3 테스트 실행 수행으로 주로 돌아가서 설명할 것입니다. DevOps 환경에서 비기능 테스트의 실행, 테스트 결함 보고, 그리고 테스트 로그 작성에 대한 프랙티스는 DevOps 환경에서 기능 테스트를 위한 프랙티스와 기본적으로 동일합니다. 이는 전통적인 프로젝트에서보다 훨씬 적은 문서 집중도로 수행될 것입니다. 종종 상세한 테스트 절차와 테스트 로그가 생성되지 않습니다. 비기능적 측면은 또한 배포 후에 테스트/측정되며, 여기서 서비스 수준 지표(SLI)와 서비스 수준 목표(SLO)가 이를 관리하는 일반적인 방법입니다.

비기능 테스트 실행은 CI/CD 파이프라인의 필수적인 부분이며, 반복 계획 동안 정의된 우선순위에 따라 수행됩니다. 일부 테스트는 문서화된 테스트 절차를 사용하여 실행될 수 있지만, 일반적으로 많은 비기능 테스트도 탐색적 및 세션 기반 테스트를 프레임워크로 사용하여 실행될 것입니다. 반복적인 개발에서는 회귀 테스트를 조직하고 구조화할 필요성이 증가합니다. 이는 자동화된 회귀 테스트와 지원 도구를 사용하여 수행될 것입니다. 회귀 테스트는 당연히 테스트가 중요하다고 식별된 시스템의 비기능적 측면에도 적용됩니다.

테스팅 중에 발견된 비기능적 결함은 팀에 의해 로깅되고 보고될 수 있습니다. 발견된 모든 결함을 실제로 로깅해야 하는지에 대해 DevOps 프로젝트 내에서는 일반적으로 논의가 있습니다. 일부 팀은 반복(iteration)을 벗어나는 결함만 로깅하고, 일부는 당일 수정할 수 없는 경우에만 로깅하며, 일부는 높은 우선순위 결함만

로깅합니다. 발견된 모든 결함이 로깅되지 않는다면, 어떤 결함을 로깅해야 하고 어떤 결함을 로깅해서는 안 되는지를 결정하는 기준이 있어야 합니다. 때로는 결함이 백로그 항목의 작업 보드에 스티커로 또는 별도의 작업으로 로깅되어, 스토리와 그 작업을 완료하지 못하도록 차단하고 있음을 시각화합니다. 일부는 결함 관리 또는 결함 추적 도구와 같은 도구를 사용하여 발견된 결함을 로깅하고 문서화하기로 선택합니다. 그러한 도구는 가능한 한 린(lean)하게 사용되어야 하며, 부가적인 가치가 없거나 너무 적은 비기능 테스트 결함 보고 요소 제출을 강요해서는 안 됩니다. 비기능 요구 사항이 프로덕션에서 측정되는 경우, 저하 지표(SLI)가 SLO를 충족하지 못함)는 이전 버전으로의 자동 롤백을 촉발하여, 전반적인 시스템 성능을 유지합니다. 테스트를 거친 항목이 정의된 인수 기준을 충족하고 실제로 "완료(done)"로 분류될 수 있는지 판단하기 위해 비기능 테스트 실행 중에 데이터를 로깅하는 것은 DevOps 팀에서도 좋은 프랙티스로 간주됩니다. 로깅된 정보는 팀과 이해관계자가 수행된 모든 테스트의 현재 상태를 쉽게 이해할 수 있는 방식으로 일부 형태의 상태 관리 도구(예: 테스트 관리 도구, 작업 관리 도구, 작업 보드)에 기록 및/또는 요약되어야 합니다.

3.5 프로세스 영역 3.5 동료 검토(Peer Reviews)

동료 검토 프로세스 영역의 목적은 작업 산출물이 명시된 요구 사항을 충족하는지 확인하고, 선택된 작업 산출물에서 결함을 조기에 효율적으로 제거하는 것입니다. 중요한 부수적 효과는 작업 산출물에 대한 더 나은 이해와 예방될 수 있는 결함에 대한 이해를 높이는 것입니다.

3.5.1 SG1 동료 검토 접근 방식 수립(Establish a Peer Review Approach)

3.5.1 SG1 동료 검토 접근 방식 수립 (Establish a Peer Review Approach)

DevOps 팀은 일반적으로 사람들이 제품에 대한 피드백을 제공하기 위해 만나는 단일하게 정의된 시간을 갖는다는 의미의 공식적인 동료 검토(formal peer reviews)를 수행하지는 않습니다. 그러나 그들은 개발 전반에 걸쳐 지속적인 비공식 동료 검토를 수행함으로써 동료 검토의 의도를 달성합니다. 이는 많은 DevOps 조직에서 일반적인 프랙티스입니다. DevOps는 흐름(flow)에 더욱 중점을 두고 소프트웨어 개발에서 린(lean) 프랙티스를 구현하기 때문에, 모든 검토 프랙티스는 활동 흐름에서 병목 현상이나 지연을 만들지 않고 이 빠른 속도의 개발을 지원합니다. 더 작은 작업 항목을 검토하는 것이 일반적입니다. 예를 들어 몇 줄의 코드에 대한 빈번한 코드 검토를 수행하거나, 백로그 항목에 대한 인수 기준을 협의하여 정의하는 것입니다. 검토 프랙티스는 프로세스에서 매우 중요하기 때문에, 예를 들어 모든 풀 리퀘스트(pull request)에서 검토를 의무화하거나 준비 완료 정의(Definition of Ready) 준수를 의무화함으로써, 피할 수 없게 만들어집니다. 이 TMMi 목표는 프로젝트 내에서 동료 검토 접근 방식을 수립하기 위한 프랙티스를 다룹니다. 검토 접근 방식은 검토 활동이 어떻게, 어디서, 언제 이루어져야 하는지, 그리고 해당 활동이 공식적인지 비공식적인지를 정의합니다. 동료 검토 접근 방식을 수립하는 것은 DevOps 프로젝트에도 적용 가능합니다. 검토 접근 방식의 가능한 한 많은 부분이 도구 내에 구현되어 피할 수 없게 설정됩니다. 예를 들어, 작업 흐름을 관리하는 작업 보드 단계 사이에서, 풀 리퀘스트에서, 또는 CI/CD 파이프라인의 품질 게이트로서 구현될 수 있습니다.

DevOps 프로젝트에서 일반적으로 수행되는 동료 검토의 예시는 다음과 같습니다:

- 반복(iteration) 기간 동안 정기적으로 팀 및 비즈니스 이해관계자와 함께 사용자 스토리에 대한 백로그 다듬기 세션을 갖는 것.

- 개발 중인 작업 산출물, 예: 코드 또는 테스트 등에 대해 논의하고 피드백을 제공하기 위해 다른 팀 구성원들과 일일 회의를 갖는 것.
- 고객에게 제품을 일찍 그리고 자주 시연하는 것, 적어도 반복 검토 동안 반복이 끝날 때 시연하는 것.
- 인프라 코드화(Infrastructure-as-Code)의 일부로서 사용자 스토리, 풀 리퀘스트를 통한 코드 변경, 스크립트 및 구성 파일과 같은 작업 산출물에 대한 검토.
- 반복 기간 동안 모든 활동에 대해 짝을 지어 작업(pair working)하는 것.

미흡한 명세(Poor specifications)는 종종 프로젝트 실패의 주된 원인이 됩니다. 명세 문제는 사용자들의 진정한 필요에 대한 통찰력 부족, 시스템에 대한 전체 비전의 부재, 중복되거나 모순되는 기능, 그리고 기타 의사소통 오류로 인해 발생할 수 있습니다. DevOps 개발에서는 사용자 스토리가 비즈니스 담당자, 개발자, 테스터의 관점에서 요구 사항을 포착하기 위해 작성됩니다. 순차적 개발에서는 기능에 대한 이러한 공유된 비전이 요구 사항 작성 후 공식적인 검토를 통해 달성됩니다. DevOps 개발에서는 이러한 공유된 비전이 요구 사항이 수립되는 동안 빈번한 비공식 검토를 통해 달성됩니다. 이러한 비공식 검토 세션은 종종 백로그 정제 세션(backlog refinement sessions)이라고 불립니다. 다듬기 회의 동안 팀과 비즈니스 담당자는 검토 기법을 사용하여 공통의 이해를 확립하고, 구현에 필요한 상세 수준을 찾고, 미해결된 이슈를 명확히 합니다.

팀이 개발 중인 작업 산출물에 대해 거의 지속적으로 검토하는 것은 전체 팀 접근 방식(whole team approach)의 일부입니다. 이러한 검토는 결함을 조기에 식별하고 개선을 위한 기회를 포착할 의도를 가지고 수행됩니다. XP(익스트림 프로그래밍) 또는 짝 프로그래밍(pairing)과 같이 DevOps에서 적용되는 방법 및 기법들 역시 팀을 위한 피드백 루프를 생성하는 핵심 프랙티스로서 동료 검토(peer reviews)를 포함합니다. 물론 높은 복잡성 또는 위험의 경우 팀은 세미 공식적이거나 공식적인 검토 기법, 예를 들어 인스펙션(inspection)을 적용하도록 선택할 수 있습니다. 이러한 경우에는 보다 공식적인 검토에 노력을 기울이고 보다 규율 있는 작업 방식을 적용하는 것에 대한 명확한 근거가 있습니다. 검토 접근 방식의 일부로서, 언제 더 공식적인 검토 기법을 적용할지를 명시하는 기준이 정의될 수 있습니다.

요구사항 엔지니어링이 비공식 검토, 워크스루, 인스펙션 또는 관점 기반 읽기(perspective-based reading)와 같은 방법을 통해 요구사항 검증(validation)을 강조하는 반면, 애자일 및 DevOps 방법은 찾고 빠른 피드백을 통해 요구 사항을 검증하여 가치 있는 제품 증분(product increments)을 신속하게 구현하는 데 주력합니다. 프로토타입 또는 프로덕션에 배포된 통합 제품 증분 형태로 빠른 결과를 보여줌으로써 조기 공식 검증의 필요성은 줄어듭니다. 만약 증분이 이해관계자의 요구 사항을 완전히 충족하지 못하면, 그 차이(delta)는 새로운 요구 사항 형태로 제품 백로그에 다시 추가되어 다른 모든 백로그 항목과 함께 우선순위가 지정됩니다. 반복(iteration) 검토 중 실제로 무엇이 구축되었는지 시연(demonstration)하는 것은 구체적인 것을 중심으로 검증 기반 대화를 활성화하는 매우 효율적인 방법입니다. 실제로 작동하는 방식을 보고, 배포된 기능이 최종 사용자 행동에 미치는 영향을 이해하는 것만큼 대화에 초점을 제공하는 것은 없습니다. 시연은 반복 검토 동안 수행되는 활동입니다. 시스템에 통합된 기능이 이해관계자에게 시연되고 논의되며, 논의로부터 얻은 새로운 학습 내용을 반영하기 위해 제품 백로그 또는 릴리스 계획에 필요한 조정이 이루어집니다.

동료 검토 기준은 자동화된 방식으로 정의되거나, 워크플로우 내의 체크리스트로서 정의될 수 있습니다. 코드 검토의 경우, 의무적인 단위 테스트 실행, 코드 복잡도 측정, 코드 스타일 가이드 준수와 같은 진입 기준이 정의됩니다. 이는 검토할 코드가 진입 기준을 준수하도록 보장합니다. 종료 기준으로, 코드는 일반적으로 팀의

다수 구성원의 검토를 통과해야 합니다. 만약 통과하지 못하면, 코드를 업데이트한 후 검토를 반복해야 합니다. 이는 일반적으로 해당 코드가 병합될 수 없음을 의미합니다.

백로그 항목의 경우, 진입 기준(entry criteria)은 종종 공식적인 검토에만 적용 가능하며, 따라서 DevOps 프로젝트 내의 백로그 정제 세션(refinement sessions)에서는 관련성이 없을 가능성이 높습니다. 그러나 종료 기준(exit criteria)의 사용은 적용 가능하며 부가적인 가치를 가집니다. INVEST 는 사용자 스토리를 검토하고 업데이트할 때 DevOps 프로젝트에서 일반적으로 언급되는 종료 기준 세트의 예시입니다. INVEST 는 좋은 사용자 스토리를 구성하는 다음 요구 사항들을 포괄하는 약어입니다:

- Independent (독립적)
- Negotiable (협상 가능)
- Valuable (가치 있는)
- Estimable (추정 가능)
- Small (작은)
- Testable (테스트 가능)



그림 6: INVEST 사용자 스토리 기준

물론, 다른 품질 관련 종료 기준도 존재하며 유익하게 사용될 수 있습니다.

검토 종료 기준은 다양한 작업 산출물에 대한 완료 정의(Definition-of-Done)에 정의됩니다. 기준을 충족하지 못하는 항목은 되돌려져(moved back) 다음 단계로 진행할 수 없습니다.

3.5.2 SG2 동료 검토 수행(Perform Peer Reviews)

물론, 다른 모든 접근 방식과 마찬가지로, 동료 검토 접근 방식은 정의되고 합의된 것으로 존재하는 것뿐만 아니라, 준수되어야 합니다. 팀은 반복(iteration) 기간 동안 백로그 다듬기 세션에 상당한 시간을 할애하고, 일상적인 루틴의 일부로 비공식적인 (그리고 가능하면 공식적인) 검토를 적용하며, 이해관계자 시연(demos)을 정기적으로 가져야 합니다. 이해관계자/고객 시연은 적어도 각 반복의 끝에 있을 것으로 예상됩니다.

테스터의 검토 참여 및 기여

DevOps 팀의 일원인 테스터가 검토 세션에 참여하는 것은 중요합니다. 이는 특히 백로그 다듬기 세션 및 협업적 설계와 같이 반복(iteration) 전반에 걸쳐 테스트의 기초로 사용되는 작업 산출물이 논의되고 검토되는 세션에서 특히 그러합니다.

일반적으로 테스터의 고유한 관점은 누락된 세부 사항이나 비기능 요구 사항을 식별하여 사용자 스토리를 개선할 것입니다. 테스터는 특정 사용자 스토리에 대한 인수 기준의 식별 및 정의를 특히 지원할 수 있습니다.

BDD(행동 주도 개발) 및 ATTD(인수 테스트 주도 개발)와 같은 기법을 사용하여 인수 테스트 형식으로 인수 기준을 정의할 수 있으며, 이를 통해 사용자 스토리가 테스트 가능성 기준을 충족하는지 판단할 수 있습니다.

이러한 방식으로 테스터는 비즈니스 담당자에게 사용자 스토리에 대한 개방형 "만약 ~라면?" 질문을 하고, 사용자 스토리를 테스트할 방법을 제안하며, 인수 기준을 확인하여 기여합니다.

특정 프랙티스 2.3 동료 검토 데이터 분석은 공식적인 검토와 특히 관련이 있습니다. 공식적인 검토에는 많은 노력이 투입되므로, 그 노력이 효율적이고 효과적으로 사용되도록 보장하기 위해 동료 검토 데이터가 수집되고 팀에 전달되어 검토 프로세스를 학습하고 조정하는 데 사용됩니다.

언급된 바와 같이, 공식적인 검토는 DevOps 프로젝트에서 덜 일반적입니다. 공식적인 검토의 필수적인 부분인 데이터 수집은 비공식적인 검토에서는 훨씬 덜 일반적입니다. 동료 검토를 수행할 때 데이터를 수집할지 여부를 결정하는 맥락에서, 다음 질문들을 던져야 합니다:

- 이 데이터가 수집된다면 누가 사용할 것인가?
- 이 데이터가 우리의 비즈니스 목표와 어떻게 관련되는가?

만약 아무도 그 데이터를 의미 있게 사용하지 않을 것이라면, 귀중한 리소스를 수집하는 데 낭비하지 마십시오. 하지만 DevOps에서는 일반적으로 도구를 사용하여 검토 결과와 데이터를 수집하고 대시보드에 정보로 제공함으로써 훨씬 더 효율적으로 만듭니다. 또한, 회고 회의 동안 팀이 검토 데이터와 발견 사항을 분석하여 근본 원인을 이해하고 개선 기회를 식별할 때, 이는 매우 의미 있게 됩니다.

DevOps 관련 용어집

블루-그린배포	두 개의 동일한 프로덕션 환경(하나는 활성 상태, 다른 하나는 유틸 상태)을 유지하여, 유틸 환경에서 새 버전의 소프트웨어를 테스트한 후, 검증이 완료되면 모든 트래픽을 활성 환경에서 전환하는 배포 전략.
카나리 테스트	변경 사항을 소규모 사용자 그룹에게 릴리스하여 검증을 허용한 후, 추가적으로 점진적인 배포를 진행하는 테스트 접근 방식.
변경 실패율	배포된 변경 사항 중 실패로 이어지는 비율.
변경 리드 타임	커밋(commit)이 프로덕션에 도달하는 데 걸리는 시간.
CI/CD 파이프라인	새로운 소프트웨어 버전을 전달하는 데 필요한 일련의 단계.
컨테이너화	애플리케이션과 그 종속성을 컨테이너라고 불리는 단일 실행 단위로 패키징하는 경량 가상화 형태.
지속적인 전달 (CD)	코드 변경 사항이 자동으로 빌드, 테스트되며, 프로덕션에 배포 가능한 상태가 되는 자동화된 소프트웨어 개발 절차.
지속적인 배포 (CD)	모든 테스트를 통과한 코드 변경 사항이 수동 개입 없이 자동으로 프로덕션에 배포되는 자동화된 소프트웨어 릴리스 절차.
지속적인 모니터링	사용자 및 시스템 행동을 실시간으로 이해하기 위한 성능 지표의 자동화된 수집.
교차 기능 DevOps 팀	서로 다르지만 중복되는 지식, 기술 및 역량 세트를 가진 사람들이 공동의 목표를 향해 함께 일하는 그룹.
다크 런치	새로운 기능 또는 제품이 공개 발표 없이 릴리스되어, 팀이 프로덕션에서 성능을 테스트하고 모니터링할 수 있도록 하는 배포 전략.
배포 빈도	소프트웨어가 프로덕션에 릴리스되는 빈도.
데브옵스	솔루션을 효과적으로 개발하고 운영하는 데 필요한 통합, 자동화 및 협업을 지원하는 사고방식, 문화 및 일련의 기술적 프랙티스.
배포 실패 복구 시간	배포 실패 후 정상적인 운영을 복원하는 데 필요한 기간.
기능 토글	코드를 변경하지 않고 시스템 동작을 수정하는 메커니즘.
피드백	제품, 서비스 및 프로세스를 개선하기 위한 정보의 지속적인 교환.
흐름	관련 가치 흐름 내에서 작업 산출물이 단계별로 원활하고 선형적으로 빠르게 이동하는 것.
인프라코드화(IaC)	기계가 읽을 수 있는 구성 파일을 사용하여 IT 인프라를 관리하고 프로비저닝하며, 소프트웨어 개발 프랙티스를 적용하는 프랙티스.
관찰 가능성	모니터링, 문제 해결 및 최적화를 가능하게 하기 위해 구성 요소 또는 시스템의 외부 출력을 분석하여 그 상태를 이해하는 능력.
풀 리퀘스트 (PR)	코드베이스에 변경 사항이 발생했음을 팀 구성원에게 알리고, 이러한 변경 사항을 검토, 논의하여 메인 코드 저장소에 병합하도록 제안하는 메커니즘.

서비스 수준 지표 (SLI)	서비스 신뢰성을 정량화할 수 있는 측정치.
서비스 수준 목표 (SLO)	서비스 수준 지표에 대한 신뢰성 목표.
사이트 신뢰성 엔지니어링 (SRE)	자동화된 피드백 루프를 사용하여 서비스 신뢰성과 새로운 기능 개발 속도의 균형을 맞추면서, 인프라 및 운영에 소프트웨어 개발 프랙티스와 원칙을 적용하는 접근 방식.
가치 흐름	정보와 자재의 흐름을 포함하여 제품 또는 서비스를 고객에게 제공하는 데 필요한 종단 간 활동의 순서.
가상화	다른 소프트웨어가 실행되는 소프트웨어 및/또는 하드웨어를 시뮬레이션하는 것.

References

- [Baah] A. Baah (2017), Agile quality assurance, Bookbaby
- [Beyer] B. Beyer, C. Jones, J. Petoff and N.R. Murphy (2016). Site Reliability Engineering: How Google Runs Production Systems, 1st. O'Reilly Media, Inc.
- [Cohn] M. Cohn (2009), Succeeding with Agile: Software Development using Scrum, Addison-Wesley
- [DORA16] A. Brown, N. Forsgren, J. Humble, N. Kersten and G. Kim (2016), 2016 State of DevOps Report, Puppet Labs, DORA (DevOps Research & Assessment)
- [DORA24] DORA [2024], DORA's software delivery metrics: the four keys, <https://dora.dev/guides/dora-metrics-four-keys/>, DORA (DevOps Research & Assessment)
- [Galen] R. Galen (2015), Three Pillars of Agile Quality and Testing: Achieving Balanced Results in your Journey Towards Agile Quality, RGCG, LLC
- [ISO25010] ISO/IEC 25010 (2023), Systems and Software Engineering – Systems and Software Quality Requirements and Evaluation (SQuaRE) – Product quality model, International Organization for Standardization
- [McMahon] Paul. E. McMahon (2011), Integrating CMMI with Agile Development – Case Studies and Proven Techniques for Faster Performance Improvement, Addison Wesley
- [Veenendaal12] E. van Veenendaal (2012), PRISMA: Product Risk Assessment for Agile Projects, in: Testing Experience, Issue 04/12, December 2012
- [Veenendaal24] Erik van Veenendaal, Rex Black, and Dorothy Graham (2024), Foundations of Software Testing - ISTQB Certification (5th edition), Cengage
- [Wake] B. Wake (2003), INVEST in Good Stories, and SMART Tasks, xp123.com/articles/invest-in-good-stories-and-smart-tasks