**백서** | 2016년 3월

# 데브옵스와 테스터

테스트 권위자이자 컨설턴트인 Paul Gerrard가 일련의 문서를 통해 테스트와 관련한 여러 주제를 다룹니다. 여기에서는 테스터 및 테스트 관점에서 바라본 데브옵스의 채택에 대해 이야기합니다. 데브옵스는 조직이 소프트웨어를 더 자주, 보다 나은 품질로 제공하기 위해 사용하는 전체적인 접근법의 일부입니다. 성공적인 데브옵스 구현의 가장 확실한 결과물은 소프트웨어 변경이 아이디어에서 프로덕션으로 전환하는 데 걸리는 시간의 단축입니다.

Paul Gerrard
Gerrard Consulting

후원:
ca technologies

# 테스터에게 데브옵스의 의미

## 배경

이 문서에서는 테스터 및 테스트 관점에서 바라본 데브옵스의 채택에 대해 이야기합니다. 데브옵스로의 전환 속도가 점점 가속화되고 있습니다. 업계에서 이뤄진 다른 여러 전환과 비슷하게, 채택의 속도는 전환 자체의 정의보다 빠르게 가속화됩니다. 데브옵스는 아직 명확하게 정의되지 않았으며 문화의 미묘한 차이, 새롭게 태어나는 신기술의 가능성, 그리고 다양한 사례 연구(대부분 성공적인) 등은 지금의 문제가 아직 널리 논의 중임을 말해 줍니다.[1]

누구와 이야기하는지에 따라 데브옵스는 문제에 대한 해결책이 될 수도 있고, 목표 그 자체가 될 수 있습니다. 일부 기업에 있어 목표는 "디지털화"이며, 데브옵스는 고품질의 제품을 수시로 딜리버리하기 위한 전반적인 접근법의 일부로 여겨집니다. 이 문서는 이러한 맥락에서 전개됩니다. 하지만 데브옵스 관련 기술 및 서비스의 마케팅에서는 이 목표가 모호할 수 있습니다. 성공에 꼭 필요한 문화적 변화(또는 더 구체적으로는 행동의 변화)라는 과제가 과소평가되는 경우가 많습니다.

또한, 이 문서에서는 데브옵스에 관여하고 데브옵스의 영향을 받는 테스터가 전체 개념에 익숙하지 않다고 가정합니다. 이 문서에서는 이러한 테스터를 위해 데브옵스를 소개하고 테스트 프랙티스에 미치는 영향을 소개합니다. 데브옵스 경험이 많은 사람에게도 유용한 정보가 될 것입니다. 테스터가 아닌 독자라도 최소한 테스터의 관점을 이해할 계기가 될 것입니다.

## 처음 접하는 사람을 위해: 데브옵스란 무엇인가?

단순히 말하자면 데브옵스는 개발 팀과 시스템 운영 팀이 더 긴밀하게 공동으로 작업하는 것을 가리킵니다. 개발자는 소스 코드 커밋부터 프로덕션 운영에 이르는 딜리버리 파이프라인에 운영 팀 활동 중 일부를 수용하고 자동화합니다. 또한 운영 팀은 개발자의 활동에 대한 더 높은 가시성을 가지고 어느 정도의 영향력을 발휘합니다. 이렇게 하는 주된 이유는 소프트웨어 배포 및 구현 속도를 높이려는 것입니다. 운영과 개발을 애자일 팀에 더 긴밀하게 결합하는 것을 "애자일 운영"이라고 할 수 있습니다.

성공적인 데브옵스 구현의 가장 확실한 결과물은 소프트웨어 변경이 아이디어에서 실제 프로덕션 운영으로 전환하는 데 걸리는 시간의 단축입니다. 개발자가 소프트웨어 변경이 "완료"되었다고 말하면, 여러 분야의 자동화를 통해 프로덕션 사용으로의 전환이 수행됩니다. 자동화 도구와 프로세스는 시스템 구성, 빌드 프로세스, 테스트, 테스트를 위한 배포, 스테이징 및 프로덕션 환경, 배포 후 모니터링, 평가, 운영에서 사용됩니다.

## 그렇다면 데브옵스가 단순히 도구와 관련된 것일까요?

어느 수준에서 보자면 데브옵스의 목표는 자동화를 통해 딜리버리 파이프라인의 병목 지점을 제거하는 것입니다. 하지만 단계별 프로세스의 자동화에도 여전히 통제가 필요합니다. 자동화된 프로세스 대부분은 실제로 독립적이지 않습니다. 즉, 유지 관리 또는 예외 처리에서 사람의 개입 없이는 작업을 완료할 수 없습니다. 완전 자동화된 데브옵스 프로세스는 인간이라는 요소를 고려하지 않으면 의미가 없습니다. 도구가 수많은 일을 처리하기는 하지만 도구 작동 프로세스를 실행하는 주체는 사람이며, 사람이 없으면 실패합니다.

그렇다면 데브옵스가 단순히 도구의 도움을 받아 개발 팀과 운영 팀이 서로 긴밀히 작업하는 것을 의미할까요?

그렇지 않습니다. 자동화된 프로세스 사이의 전환에는 다른 프로세스가 관련되는 경우가 많습니다(일반적으로 여러 종류의 테스트). 자동화된 테스트는 개발자와 테스터가 작성해야 합니다. 이러한 테스트의 결과물은 파이프라인의 단계 간 전환을 위한 다른 프로세스 또는 사람들에게 충분한 정보를 제공하는 데 초점이 맞춰집니다. 테스트를 수행하는 개발자와 테스터는 데브옵스 프로세스가 성공적이고 안정적인 딜리버리를 제공한다는 점을 보장할 수 있어야 합니다.

"머리가 복잡하네요. 데브옵스란 실제로 무엇인가요?"라고 묻는다면 새롭게 부각되고 있는 원리라고 답할 수 있습니다. 이 질문은 여기[1]에 있는 훌륭한 게시물에서 제기되고 논의되었습니다. 이 논의는 본 문서를 작성하기 몇 주 전에 이뤄졌습니다. 이렇게 데브옵스의 정의는 아직도 정립 중이라고 할 수 있으며, 아마도 영원히 정립되지 않을 수도 있습니다.

테스터에게 데브옵스란 어떤 의미일까요? 이는 아직 "정답"이 없으며 새롭게 부상 중인 데브옵스 조직에서 여러분이 가지는 역할도 아직 확립되지 않았음을 의미합니다. 여러분이 기여할 수 있는 두 가지 주요 역할은 다음과 같습니다.

1. 고통을 주는 요소에 주의를 기울이고 그 고통을 최소화하기 위한 작업을 해야 합니다.
2. 데브옵스 프로세스에 가치를 더할 수 있는 기회 및 개입을 파악해야 합니다.

데브옵스로 유도하는 원동력을 가장 잘 설명하는 격언은 "고통스러운 일은 더 자주 하라"입니다. 진부한 표현일 수 있지만 이 문서에서는 데브옵스 테스트 프랙티스의 구현 및 개선을 위한 맥락으로 사용하겠습니다.

## 고통스러운 일이라면 더 자주 하라

특정 작업을 할 때 경험하는 어려움이나 고통은 우리에게 부정적인 영향을 미칩니다. 싫어하는 작업이라면 이를 미루게 되는 경우가 많습니다. 그러다 결국 작업을 시작하게 되면 그 고통은 더 커집니다. 이는 치과 가기, 창고 청소, 소프트웨어 통합, 테스트 등의 경우도 마찬가지입니다. 경험상 이러한 작업을 수행하는 빈도가 낮을수록 해당 작업을 실제로 수행할 때의 고통도 커집니다. Martin Fowler는 특정한 작업을 자주 또는 지속적으로 수행할 때 고통이 줄어드는 이유를 이렇게 설명합니다.[2]

첫 번째로, 크고 복잡한 작업일수록 계획, 관리 및 제어가 더 어렵습니다. 큰 작업을 쪼개면 수행하기가 쉬워지고 위험도 줄어들며 무언가 잘못되었을 때 되돌리기도 더 용이합니다. 두 번째로, 여러 개의 작업(이 예에서는 테스트)은 피드백을 제공합니다. 이러한 피드백을 자주 그리고 초기 단계에 받을 수 있으면 시간을 더 낭비하기 전에 문제를 빠르게 해결할 수 있습니다. 세 번째로, 어떤 활동이든 더 자주 수행하게 되면 더 잘 수행하게 됩니다. 효율적으로 수행하는 방법을 배우게 되는 것입니다. 또한 어떤 방법으로든 자동화할 기회도 찾을 수 있게 됩니다.

테스터의 관점에서 볼 때 이 격언은 테스트 프로세스의 자동화를 더 진지하게 받아들이도록 합니다. 수동 개입이 존재한다면(일반적으로 데브옵스 프로세스 중 자동화된 단계 사이에) 이 지점이 바로 고통의 지점이자, 병목 지점, 지연의 원인, 프로세스 중 신뢰성이 낮고 오류가 발생하기 쉬운 지점이 될 것입니다. 수동 테스트는 고통입니다. 독자 여러분도 예비 테스트를 더 선호할 수 있습니다. 또한 자동화로는 결코 찾을 수 없는 복잡한 버그는 사람만 찾을 수 있고, 테스터만이 재해 발생을 막을 수 있는 믿을 만한 사람이라는 생각에 두려움을 갖고 있을 수도 있습니다.

테스터인 여러분은 개발자나 자동화가 테스트 작업을 제대로 해낼 것이라고 신뢰하기가 고통스러울 수 있습니다. 고통스럽다면 더 자주 해야 합니다.

## 테스트, 자동화 및 신뢰

검사와 테스트의 의미에 대한 많은 논의가 존재하며[3] 테스터, 검사 및 자동화에 부여할 수 있는 신뢰도에도 많은 논의가 존재합니다.[4, 5]

자동화된 검사를 완벽하게 신뢰할 수 있다는 말이 아닙니다. 그 보다 더 정교한 것이 필요하다는 것만은 확실합니다. 하지만 이 문서의 목적상 최소한 테스트와 테스트 실행 활동을 네 가지 구성 요소로 구분할 수 있습니다.

1. 개발자가 구성 요소 수준 검사 및 지속적인 통합 프로세스의 일부로 자동화할 수 있는 검사
2. API 수준, 링크 또는 엔드-투-엔드 트랜잭션을 실시하기 위해 자동화할 수 있는(일반적으로 시스템 테스터가) 검사
3. 브라우저, 운영 체제, 플랫폼 간의 호환성을 입증하기 위해 호환성 검사를 수행할 수 있는 테스트
4. 사람만이 수행할 수 있는 테스트

물론 환경이 저마다 다르기 때문에 이 문서에서는 이렇게 구분하는 방법을 몇 가지만 제안할 수 있습니다. 이 문서에 더 적합한 질문은 "테스터가 후반 단계의 수동 검사를 없애는 방법은 무엇인가?" 입니다. 필자는 이전에 후반 단계의 수동 검사 제거에 대해 논의한 적 있습니다.[6] 여기에는 적극적인 노력과 신뢰가 필요합니다.

다음 사항에 초점을 맞춰야 합니다.

1. 가능할 때마다 구성 요소 수준에서 수행 가능한 수동 검사를 개발자에게 맡겨야 합니다. 테스터는 역할 분담 또는 화이트보드 세션에서 이러한 테스트를 제안할 수 있습니다. 이를 직접 작성한 다음 지속적인 통합 제어에 이를 포함해야 할 수 있습니다.
2. 엔드-투-엔드 또는 사용자 인터페이스 테스트에는 자동화가 필요할 수 있습니다. 이러한 테스트는 실행이 느리고 민감하며 유지 관리가 필요한 경우가 많기 때문에 최소화해야 합니다. 모든 코드 체크인에서 실행해야 할지 아니면 향후의 더 크고 빈도가 더 낮은 릴리스에서만 사용하도록 예약할 수 있는지를 고려합니다.
3. 아직 릴리스 후보로 통합되지 않은 구성 요소에 대해 수동으로만 실행할 수 있는 테스트는 무엇인가? 개발자와의 역할 분담 세션에서 수동 테스트를 수행할 수 있는가? 이 테스트의 대안이 있는가? 스토리 보딩, BDD 스타일 프로토타이핑이 도움이 될 수 있는가? 모형(mock-up) 또는 와이어 프레임에 대해 UI 검사를 수행할 수 있는가?
4. 회귀 용도로 보존해야 하며 자동화의 대상인 검사와 수동으로 한 번만 실행해야 하는 검사는 무엇인가?

앞서 신뢰의 개념을 언급한 적 있습니다. 이를 바라보는 다른 방법은 후반부의 수동 테스트가 전혀 없다면 시스템을 어떻게 높은 신뢰도로 테스트할 수 있는지를 생각해보는 것입니다. 모든 테스트가 도구에 의해 수행되는 환경을 예로 들어보겠습니다. 개발자가 테스트를 제대로 수행할지에 대한 믿음이 없다는 것이 가장 큰 걱정거리인가요? 테스트에 대한 관점을 바꾸면(이전 문서에서 제안한 것처럼) 불신이 줄어듭니다. 테스터인 여러분이 위험을 식별하고 평가하는 개척자 역할을 하여 테스트를 선택하고 이를 개발 및 자동화에 통합한다면 이러한 걱정거리를 최소화할 수 있습니다.

여러분이 품질의 감시자, 최후의 방어선, 유일하게 책임감을 갖는 사람이라는 생각을 버려야 합니다. 그 대신 스스로를 비전 제시자, 위험 식별자, 위험 관리자, 개척자, 관리자, 코치/멘토라고 생각해야 합니다.

### 실시, 모니터링 및 구현

후반부 수동 검사에 대한 의존도를 축소 또는 제거하려는 좋은 의도에도 불구하고 버그는 생겨납니다. 소프트웨어가 프로덕션으로 릴리스되면 문제가 발생합니다. 운영의 관점에서 볼 때 데브옵스의 핵심 원리는 더 심도 깊은 모니터링입니다.

구성 요소와 애플리케이션의 단순한 트랜잭션부터 통합, 메시징 및 인프라까지 모든 계층에서 모니터링하는 것입니다. 모니터링의 한 가지 목표는 사용자가 실패의 영향을 경험하기 전에 이를 경고하는 것입니다. 이는 희망 사항일 수 있지만 궁극적인 목표입니다.

프로덕션에서 문제가 발생한 경우 해야 할 일은 모니터링에서 도출된 분석을 사용하여 원인을 추적 및 해결하는 것 뿐만 아니라 향후 비슷한 문제의 재발 가능성을 줄이기 위해 자동화 또는 수동 테스트 프로세스를 다듬는 것입니다. 전체 파이프라인 프로세스에서 테스트 및 분석의 역할에 대해서는 이 문서에서 소개하고 살펴봅니다.[7]

데브옵스 프로세스에서의 자동화된 테스트를 “모니터링”이라고 할 수 있습니다. 테스트가 프로덕션의 모니터링과 결합되면, 데브옵스 프로세스 및 프로덕션 전체의 모니터링이 테스트의 범위를 확대한다고 말할 수 있습니다. 따라서 데브옵스는 테스터의 역할을 축소하지 않습니다.

## 결론

최근에 “조직에서 데브옵스를 시도하지 않아야 하는 경우는 언제인가요?”라는 질문은 받았습니다. 좋은 질문이지만 그 이면에는 데브옵스가 지속될 것인가 그리고 테스터가 이를 고려해야 하는지 여부에 대한 우려가 내포되어 있습니다. 저의 대답은 간단합니다.

개발자와 운영 담당자가 서로 대화를 나누지 않아야 할 이유가 있나요? 테스트 및 프로덕션에 안정적인 빌드와 배포를 원하지 않습니까? 더 정확하고 효율적이며 더 정보가 풍부한 파이프라인을 지원하는 최고의 기술을 원하지 않습니까? 데브옵스는 좋지만 달성하기 쉽지만은 않습니다. 당연히 문화적 변화가 수반되며 이것이 항상 쉽지는 않습니다.

테스터에게 있어 데브옵스는 프로젝트의 초기 단계에 더 큰 영향을 미치게 되며, 테스트, 정보 준비 및 의사 결정에 있어 자동화에 대해 더 진지하게 생각할 기회를 줍니다. 데브옵스는 사전 대응 방식으로 대응할 기회를 주며, 프로젝트 팀에서 더 많은 권위와 존중의 기회를 제공하기 때문에 테스터는 이를 수용해야 합니다.

# 저자 소개

Paul Gerrard는 컨설턴트, 교사, 저자, 웹마스터, 개발자, 테스터, 컨퍼런스 연설자, 로잉 코치 겸 출판인입니다. 그는 테스트 보장을 전문으로 하여 소프트웨어 테스트 및 품질 보증의 모든 측면에서 다양한 컨설팅을 수행해 왔습니다. 또한 유럽, 미국, 호주, 남아프리카 등지의 테스트 관련 컨퍼런스에서 기조 연설 및 강습을 진행했으며 이에 대한 상을 수상하기도 했습니다.

Oxford 및 Imperial College London에서 공부했으며 2010년 Eurostar European Testing excellence Award, 2013년 The European Software Testing Awards(TESTA) Lifetime Achievement Award를 수상했습니다.

2002년에는 Neil Thompson과 함께 “Risk-Based E-Business Testing”을 집필했습니다. 2009년에는 “The Tester’s Pocketbook”을 집필했습니다. 2011년에는 Susan Windsor와 “The Business Story Pocketbook”을 공동 집필했으며 2014년에는 “Lean Python”을 집필했습니다.

2014년에는 더블린 EuroSTAR 컨퍼런스에서 프로그램 의장을 맡았습니다.

현재 그는 Gerrard Consulting Limited의 사장이자 TestOpera Limited의 이사이며 Test Management Forum의 운영자입니다.

메일: paul@gerrardconsulting.com
Twitter: @paul_gerrard
웹: gerrardconsulting.com

## 자세한 정보는 CA 테크놀로지스의 **개발 및 테스트**를 참조하십시오.

**ca.com/kr을 통해 CA 테크놀로지스를 만나 보십시오.**

CA 테크놀로지스(NASDAQ: CA)는 회사가 변화를 통해 애플리케이션 경제의 기회를 잡을 수 있도록 하는 소프트웨어를 만듭니다. 소프트웨어는 모든 업종, 모든 기업의 핵심입니다. 계획부터 개발, 관리 및 보안에 이르기까지 CA는 전 세계의 회사를 도와 모바일, 프라이빗 및 퍼블릭 클라우드, 분산 및 메인프레임 환경에서 생활, 거래 및 소통의 방식을 바꾸고 있습니다. 자세한 내용은 **ca.com/kr**을 참조하십시오.

참조 자료

1. “What is DevOps,” The Agile Admin, http://theagileadmin.com/what-is-devops/
2. “Frequency Reduces Difficulty,” Martin Fowler, http://martinfowler.com/bliki/FrequencyReducesDifficulty.html
3. “Testing and Checking Refined,” James Bach, Michael Bolton, http://www.satisfice.com/blog/archives/856
4. “A New Model for Testing,” Paul Gerrard, http://dev.sp.qa/download/newModel
5. “The New Model and Testing v Checking,” Paul Gerrard, http://blog.gerrardconsulting.com/?q=node/659
6. “How to Eliminate Manual Feature Checking,” Paul Gerrard webinar, http://blog.gerrardconsulting.com/?q=node/622
7. “Thinking Big: Introducting Test Analytics,” Paul Gerrard, http://blog.gerrardconsulting.com/?q=node/630


CS200-181159_0316