Best Practices in Software Test Automation

(출처) http://azizs.blogspot.com/2005/08/best-practices-in-software-test.html

(역) techbard @TERAMAIL.com

Best Practice 란?

Best Practice 는 가이드라인이며, 어떤 것을 수행하는 최상의 방법에 대한 조언이다. 즉, 시간이 지나면서 수집된 이전 프로젝트의 경험을 바탕으로 한 것이다. 조직 내의 Best Practice 는 관리층에 의해 강제되지 않고, 밑에서 부터 위로 도출되어야 한다. 심지어 이러한 Best Practice 들은 조직마다 다를 수 있다. 하지만, 성공적인 조직과 프로젝트에 내재된 어떤 핵심적인 요인들은 존재한다. 이 아티클에서 논의되는 것들은 12 가지의 가장 중요한 프랙티스들로 당신의 조직과 프로젝트에 기능적인 테스트 자동화를 성공적으로 구현하도록 하는데 도움을 줄 수 있다.

베스트 프랙티스 1: 당신의 목적을 분명히 하라

테스트 자동화를 수행해야 하는 좋은 이유들은 매우 많다.

테스트 자동화는 시간을 절약해 주며, 테스팅을 쉽게 만들어 주며, 테스팅 커버리지를 개선시킬 수 있다. 또한 테스터들을 동기부여되게 할 수 있다. 나는 기능적인 테스트 자동화를 수행해서 얻을 수 있는 다른 이익들을 한 페이지로 나열 할 수 있다. 하지만, 당신의 조직이 이 모든 것들을 동시에 해야할 필요는 없을 것이다. 대개 서로 다른 그룹들은 그들이 원하는 자동화에 대해 서로 다른 희망과 아이디어들을 가지고 있다. 이러한 필요성들은 언급되어야 한다. 그렇지 않으면, 실망만 가지게 될 것이다.

테스트 자동화 구현 동안에 추구해야 하는 명확한 목표는 매우 중요하다. 이것은 우리에게 어떤 계획과 실행을 해야 하는지에 대한 지침을 제공한다. 최종적으로, 테스트 자동화 구현의 결과는 우리가 목표를 달성했는지 아닌지를 판단하기 위해서 원래의 목표와 비교될 것이다.

실수들은 다음과 같다. (Mistakes made...)

"테스트 자동화는 우리 테스팅 문제에 대한 해답이다"

테스트 자동화는 테스팅 프로세스를 지원하고, 따라서 테스트 팀에 가치를 제공할 수 있다. 어떤 이익들이 도출되긴 하지만, 우리는 잘못된 기대나 희망을 가지지 않고 그것들이 무엇인지를 명확하게 이해하고 알아야 한다. 자동화는 부적절한 테스트 프로세스나 효과적이지 않은 테스트 팀의 대안이 될 수 없다. 비용이 많이 드는 테스트 자동화 구현에 투자하기 전에 테스팅 문제들을 처리하라. 명백하게 자동화의 목적이 기존에 존재하는 모든 테스팅 문제에 대한 해결책은 아니다.

"우리가 테스트하는데 시간의 제약이 있기 때문에, 테스트 자동화를 사용하자"

테스트 자동화는 수동 테스팅에 비해 구현하는데 3 배에서 128 배까지 더 길어진다. 그러므로, 기대된 이익을 달성하기 위해서는 올바르게 계획되어야 한다. 만일 시간이 중요한 요인이라면, 테스트 자동화를 구현하는 매우 긴 프로세스에

투자하지 말고 테스트 프로세스에 더 많은 리소스를 투입할 것을 고려하라 (이것 또한 이익을 제한하며, 항상 해결책이 되진 않을 수 있다). 기능적인 테스트 자동화는 확실히 압박받는 테스트 프로젝트에 대한 빠른 해결책이 아니다. 이러한 욕구를 처리하기 위해서는 다양한 다른 방법들이 더 적합하다.

베스트 프랙티스 2: 테스트 자동화는 수동 테스트 프로세스를 필요로 한다

" 테스트 자동화에서의 성공은 주의깊은 계획과 디자인 작업을 필요로 한다. 그리고, 이것은 만능의 해결책이 아니다. “... 자동화된 테스팅은 수동 테스팅의 대체로 고려되어서는 안 된다. 오히려, 개선이 되어야 한다."(James Bach [pg. vii] Software Testing with Visual Test 4.0. ISBN 0-7645-8000-0 의 서문에서)

James Bach 의 말과 또 다른 많은 것들은 이 주제에 대해서 확실하다. 테스트 자동화는 올바른 테스트 프로세스나 방법론을 제공하지 않을 것이다. 이것은 방법론이나 프로세스를 강화하지도 않을 것이다. 테스트 자동화는 순전히 테스트 프로세스를 지원한다. 즉, 이것은 수동의 테스트 프로세스를 대체하지 않을 것이다.

실수는 다음과 같다.

"우리는 프로젝트에서 테스팅을 실행 할 필요가 있습니다. 툴을 구매해서, 그 테스팅을 자동화 합시다."

당신의 중요한 컨텐츠를 스크립트화 할 수 있겠지만, 올바른 수동 테스트 프로세스를 적절하게 가지고 있지 못하다면, 테스트 자동화의 이득은 결코 나타나지 않을 것이다. 이것은 당신이 틀린 프로세스를 더 많이 빠르게 수행하도록만 당신을 도울 것이다. 테스트 자동화는 순전히 테스트 프로세스 그 자체를 지원하고 개선한다. 테스트 자동화를 추가하려고 시도하기 전에 올바른 테스트 방법론이 구현되어있는지 확인하라.

"우리는 단지 현재의 테스트 프로세스에 테스트 자동화를 추가하고, 현재의 자동화된 테스트 스크립트를 시작하면 되요."

테스트 자동화는 자동 프로세스를 지원하기 위한 테스트 스크립트를 필요로 한다. 만일 나의 테스트 스크립트가 테스트 중인 시스템의 모든 중요한 로그온에 통합되어 있지 않다면, 그 자동화 스크립트는 많은 수의 수동적인 개입을 필요로할 것이다. 그러므로, 자동화되기 보다는 더 수동화가 된다. 현재의 테스트 프로세스가 바뀌어야 할 수도 있고, 개정될 수도 있고, 선정된 툴에 맞게 구현되어야 할 수도 있다. 이익을 극대화 하기 위해 그 테스트 프로세스의 리뷰와 개정은 당신의 테스트 프로세스에서 사용될 수 있는 툴 수트에서 실행되어야 한다.

베스트 프랙티스 3: 소프트웨어 개발 라이프싸이클에 더 일찍 참여할 수록, 더 많은 이익을 얻을 수 있다

우리 모두는 SDLC 단계에 테스팅이 어떻게 자신의 단계를 맞추어야 하는지 알고 있다. 프로젝트의 라이프싸이클 상에 더 일찍 테스팅이 참여하는 것이 좋다. 수많은 아티클들이 이 주제를 다루어왔지만, 모두 사실이다. 테스트 자동화가 테스트 프로세스를 지원하기 때문에, 테스트 팀은 프로젝트의 라이프싸이클에 아주 일찍 참여할 수 있다. 만일 그 테스트 팀이 테스트 자동화 툴 수트를 사용한다면, 그들은 수트에 포함된 요구사항 관리 툴이나 테스트 관리 툴을 가지고 있을 가능성이 높을 것이다. 이러한 툴들은 개발 라이프싸이클 상의 초기 단계에 여러가지 방법으로 테스트 팀을 지원할 수 있다. 심지어 코드가 한줄이라도 개발되어 있지 않더라도 말이다.

좀더 성숙한 테스트 자동화의 기법들과 방법들은 사용중인 테스팅 방법론과 밀접하게 연결된다. 테스트 라이프싸이클의 초기에 시작하고, 개발 프로젝트 라이프싸이클의 시작 지점과 함께하는 것은 최상의 결과와 최대의 ROI 를 보장하기 위해 매우 필수적이다.

실수는 다음과 같다.

"테스트 자동화 담당자는 개발자들이 첫 빌드를 릴리즈하기 전에 그 프로젝트에 포함될 수 없다."

소프트웨어 개발 프로젝트는 코드 개발과 함께 시작되지 않는다. 또한 기능적인 테스트 자동화는 개발 프로젝트이기 때문에, 베스트 프랙티스 9 에서 언급한 바와 같이, 소프트웨어 개발 프로젝트처럼 몇몇 단계를 따르며, 코딩 단계에서 시작하지 않는다. 올바른 계획과 디자인이 성공적인 테스트 자동화 구현의 성공에 필수적이다. 그리고, 테스트 라이프싸이클과 밀접하게 연관되어 진행되어야 한다.

"테스팅은 SDLC 내의 한 개의 단계이기 때문에, 테스터와 테스트 자동화는 테스팅 단계 동안 그 프로젝트의 일부분만 되면 된다."

IT 산업에서의 이 "원시 시대적인" 오해는 나쁜 소프트웨어 품질의 주요 원인중 하나임이 이미 증명되었다.

베스트 프랙티스 4: 테스트 자동화는 전업이다

인력이 자신의 개인 시간으로 테스트 자동화를 작업하라고 허용될 때나, 테스트 스케줄이 허락할 때만 허용되는 우선순위가 낮은 프로젝트이기 때문에 테스트 자동화는 부가 활동이 되고 만다. 테스트 자동화는 시간을 갖지못하고, 충분히 집중되지 못한다. 자동화된 테스팅으로 진입하기 위해서는 당신 자신만 준비되는 것 뿐 아니라, 당신의 회사와 환경도 준비되어야 한다. 이것들은 집중과 전용의 리소스를 필요로 한다.

실수는 다음과 같다.

"우리는 테스트 자동화 툴을 가지고 있다. 테스트 팀이 그것을 사용할 수 있다면 그것을 알고 있지 않을까?"

테스트 자동화에 대한 기대와 이해가 부족한 것이 명백하다. 나는 툴 구현이 성공적이지 않을 것이라는 걸 보증할 수 있다. 그리고, 테스트 팀에 거의 아무런 이익도 되지 않을 것이라는 점도. 대개 그 테스트 팀은 테스팅을 위해 코드 인수 날짜를 맞추지 못하는 개발 프로젝트를 위해 수동 테스트 작업을 해야할 것이다. 그 테스트 자동화 구현은 아마도 캡쳐-레코드-플레이백 형태가 될 것이며, 따라서 이익은 낮을 것이다.

테스트 자동화에는 참여하는 인력의 훈련을 필요로 할 것이다. 아키텍쳐를 디자인하고 구현하는데 시간이 필요하다. 기능적인 테스트 자동화 구현의 개발 속성과 관련하여, 그 팀은 새로운 기술 집합을 배워야 할 것이다. 테스트 자동화 구현의 성공을 위해서는 집중과 전담을 필요로 한다. 이러한 맥락에서, 실험하느라 임의의 시간을 배정하는 것보다는 테스트 자동화를 포기하는 편이 나을 것이다.

베스트 프랙티스 5: 올바른 제품 또는 제품 수트를 선택하라

잘못된 툴을 구매하는 것은 테스트 자동화의 주요 도전과제중 하나로 언급될 수 있다. 왜냐하면, 프로세스의 종류와 방법론 또는 조직이 어떠하든 간에, 툴이 기술적으로 비즈니스 적으로 적합하지 않다면, 사용되지 않을 것이다.

우리는 훌륭한 프로세스와 조직 또한 테스트 자동화의 필수조건임을 알고 있다. 하지만, 그 툴이 기본적인 수준에서 동작하지 않는다면, 그 툴을 사용해야 하며, 그 툴에서 이익을 얻어내야 하는 사람들은 그것을 사용하지 않게 된다.

불행히도, 극히 소수의 사람들만 테스트 툴을 구매하기 전에 적절한 조사를 수행한다. 적절한 조사에는 만족시켜야 하는 그 툴의 유저에 근거해서 툴 요구사항 집합을 정의하는 것, 후보군 툴들을 평가할 수 있는 평가 기준 세트의 개발, 그 툴을 고려해보기위해 이미 사용해 본 다른 사람들의 경험 입수 등이 포함된다.

당신이 자동화된 테스팅을 구현하기 위해서 툴을 선정할 때 당신의 장기간의 테스팅 전략과 테스트 방법론을 맞추어야 한다.

실수는 다음과 같다.
"벤더가 데모를 보여준 툴은 가치를 더해 줄 것으로 보인다. 나는 그걸 사야겠다."

기술, 프로세스, 어플리케이션, 인력의 기술 수준 그리고 조직에 대한 툴 요구사항을 정의하는 시간을 가져라. 그런 다음, 복수의 툴 벤더를 살펴보고, 당신의 상황에 적합한 것을 선정하라. 당신의 기준에 최적으로 들어맞는 툴을 가지고 개념 검증 프로젝트 (proof of concept project)를 수행하는 것은 좋은 생각이다.

"우리는 벤더로 부터 전체 툴 수트를 구매했다. 하지만, 기능 테스트 자동화 툴만 사용한다."

테스트 자동화 툴은 비싸다. 그리고, 제품 수트 내에 있는 기능적인 테스트 자동화 툴만 사용하는 것은 대개 ROI 를 정당화하기 위해서는 효과적이고 효율적이지 않다. 다른 이익들은 제품의 전체 수트를 사용해서 얻어질 수 있다. 예를 들어, 올바른 결함 관리, 테스트 상태에 대한 메트릭 리포팅 등이다. 당신이 필요로 하는 것을 구매하고, 가지고 있는 것을 충분히 활용하라.

"연간 라이센스 요금이 너무나 비싸서 우리는 이 툴 수트를 더 이상 사용할 여력이 없다."

너무나 자주 테스트 자동화 툴은 결국 "셸프웨어"가 된다. 왜냐하면, 라이센스 비용과 거기서 얻어지는 이익이 비교되기 때문이다. 왜 ROI 가 실현되지 않는가하는 이유중 한 가지는 위의 두개의 실수와 관련이 있다. 구매 계약서에 사인하기 전에 연간 라이센스 요금의 효과에 대해서 주의하라.

베스트 프랙티스 6: 상위 경영진의 동의를 이끌어내라

테스트 자동화는 매우 많은 실재적인 결과를 산출할 수 있다. 하지만, 성공을 위해서는 실재적인 동의가 필요하다. 모든 영역에서의 동의가 없는 경우 시작하지 마라. 가장 중요한 것은 경영진의 동의이다.

ST Labs 의 부사장인 Thomas R. Arnold 2 세는 다음과 같이 잘 정의하고 있다.

" 나는 먼저 다음과 같이 말하고 싶다. ... 어떤 테스트 자동화 툴도 만능이 아니다. 자동화는 시간, 노력 그리고 참여자 모두의 동의가 필요하다. 이것에는 자동화가 무엇을 할 수 있고, 무엇을 할 수 없는지에 대한 경영진의 현실적인 이해가 포함된다."

경영진의 지원은 테스트 툴의 사용을 효과적으로 지원하게 될 테스트 프로세스의 디자인과 실행, 조직 내에 자동화된 테스트 툴의 사용과 역할을 강화시키고, 테스팅 프로세스에 툴을 통합시킬 수 있는 시간을 마련하는데 필요하다.

경영진의 지원이 없다면, 전체 테스트 자동화 노력은 위험에 빠진다. 만일 경영진이 명백하고 일관되지 않게 지원하지 않는다면, 사람들은 그 툴의 사용에 흥미를 잃어버리게 될 것이다. 이것은 큰 문제인데, 특히 어떤 툴이라도 학습곡선을 극복하기 위해서는 집중이 필요하다는 점을 고려할 때는 더욱 그러하다.

아마도 경영진의 지원에서 보여지는 가장 큰 도전과제는 그 툴을 적용하는데 걸리는 시간, 노력, 학습에 비해 그 툴에 거는 더 큰 기대를 조절하는 것이다. 경영진은 툴의 성과 진행이 더딘것에 대해서 조급해져서, 그들의 지원을 다른 주도권자들에게 이동시킬 것이다.

실수는 다음과 같다.

"우리는 이 툴들이 당신의 테스팅 문제들을 해결해줄 것이라고 생각한다."

테스트 툴을 구매하기 위해 경영진과 협의할 때, 이익 뿐만 아니라 도전과제들도 제시하라. 경영진의 영향력이 다른 사람들이 자동화된 테스트 툴을 어떻게 받아들일 지를 결정한다는 점을 경영진이 이해하도록 하라.

"우리는 3 개월 전에 당신을 위해 툴을 구매해 주었습니다. 그런데 아직까지 이 프로젝트에서 어떤 이익도 보지 못하고 있습니다."

처음부터 경영진과 올바른 툴과 사람, 프로세스의 기초를 형성하는데 시간이 걸린다는 점을 의사소통하라. 경영진에 툴 도입 진행 과정에 대해 정보를 제공하고, 어떤 이슈라도 생기면 제기하라.

베스트 프랙티스 7: 프로젝트에 맞는 적절한 기술(들)을 선정하라

기능적인 테스트 자동화 영역에서 성숙도 수준이 개선됨에 따라, 우리는 테스트 자동화 구현에 들어가는 비용이 감소하며, 조직이나 프로젝트에서 도출되는 이익이 증가함을 보고 있다.

성숙도 수준의 증가는 테스트 자동화를 구현하는데 더 성숙된 기술을 우리에게 제공해왔다. 이러한 기술들은 매우 간단한 레코드&플레이백에서부터 가장 성숙한, 이른바 키워드나 액션 기반까지 다양하다. 이러한 기술들 각각은 고유의 장점과 약점을 가지고 있고, 어떤 특정한 환경에만 적합하기도 하다.

작은 데이터 변환 프로젝트에서, 이전 시스템의 소수의 룩업 데이터 테이블의 컨텐츠는 배치 변환 이후에 신규 룩업 데이터 테이블의 컨텐츠와 비교해야 한다. 만일 이것이 두 시스템에 있는 질의 기능을 통해 이루어질 수 있다면, 가장

빠른 방법은 레코드&플레이백이 될 것이다. 이것은 변환 테스트 한 번으로 끝낼 수 있다. 그리고, 이것을 수행하기 위해 특정 스크립트를 작성하는 것은 레코드&플레이백 방식보다 더 시간이 걸릴 것이다. 비용과 시간의 관점에서 레코드&플레이백 기술은 이 프로젝트에 가장 적합할 수 있다. 이것은 내가 레코드&플레이백 기술의 사용을 제안하는 소수의 예제들 중 하나이다. 이 기술의 가장 최대 이익은 더 성숙한 기술인 데이터 주도와 키워드 테스트 자동화에 비교하여 적다.

다양한 기술들이 적절하게 사용될 수 있다는 점을 아는 것이 중요하다. 만일 우리가 필요한 시점에 다양한 기술들을 사용하지 않는다면, 우리의 테스트 자동화는 우리가 의도한 이익을 제공하지 않을 것이다.

실수는 다음과 같다.
"레코드&플레이백은 우리의 필요를 충족시킨다."

이 간단한 기술의 이익은 적다. 그리고 이것의 적용은 광범위하게 잘못 사용되고 있다. 기하급수적으로 사용 편의성, 비용 절감 그리고 올바르게 구현된 경우에 시간 절약의 관점에서 더 많은 이익을 제공하는 성숙한 기술이 있다.

"우리는 그 프로젝트에 맞는 테스트 케이스를 작성해왔고, 첫 빌드의 릴리즈에 가까워지고 있다. 우리가 프로젝트가 끝나기 전에 키워드 테스트 자동화를 구현할 수 있을까?"

키워드 테스트 자동화는 아주 진보한 테스트 자동화 기술로 프로젝트에 사용된 테스팅 방법론과 밀접하게 관련이 있다. 키워드 테스트 자동화의 구현에 필요한 준비 작업에는 다른 기술보다 몇 배의 시간이 필요하다. 프레임워크가 수립되어 있어야 하며, 테스트 디자인이 방식을 지원해야 한다. 나는 이 프로젝트의 늦은 단계에서 시작부터 이 방식으로 구현하는 것을 추천하지 않는다.

베스트 프랙티스 8: 모든 테스트들을 자동화하려고 시도하지 말라

테스트 자동화는 모든 상황이나 테스트에 잘 적용되지 않는다. 어떤 테스트는 수동 실행으로 내버려두는 것이 좋다. 만일 한 테스트가 3 번이상 반복되지 않는다면, 그것은 자동화의 좋은 후보가 아닐 것이다. 이것은 수동으로 실행하는 것보다 자동화하는데 더 오래 걸린다. 자동화 테스트의 이익은 그 테스트가 얼마나 많이 반복되어야 하는지와 관련되어 있다. 예를 들어, 다중 플랫폼 상의 다중의 빌드 또는 테스트 또는 다중의 데이터 세트를 위한 동일한 테스트를 실행하는 경우가 해당된다.

수동으로 테스트되어야 하는 소프트웨어는 많은 다양한 상황에서 버그를 찾아내는데 도움이 되는 임의성을 가지고 테스트될 것이다. 하나의 소프트웨어 프로그램은 대개 다양하게 테스트되지 않기 때문에, 자동화된 테스팅은 수동 테스팅이 찾아내는 것만큼의 버그를 찾지 못할 것이다. 자동화된 소프트웨어 테스팅은 결코 수동 테스팅의 완전한 대체가 될 수 없다.

들어간 노력에 비해 ROI 가 확실하도록, 자동화하기 위해 올바른 테스트 케이스를 확인하는 것에는 올바른 분석이 필요하다. 자동화를 위해 어떤 테스트를 선정할 것인가는 그 자체로 방대한 주제이다. 하지만, 다음의 제안들이 도움이 될 것이다.

기술적인 적용성과 ROI 메트릭을 결정하기 위해 그 시스템에 맞는 테스트 케이스의 상호 섹션 상에서 POC (proof-of-concept)를 수행한다.

자동화 후보를 선정하기 위해 각 테스트 방법에 대한 ROI 분석을 수행한다.

어떤 테스트가 먼저 자동화되어야 하는지 우선순위를 결정하기 위해 자동화 후보에 대해 위험 분석을 수행한다.

만일 기능적인 테스트 자동화를 위해 선정된 테스트가 확인되고 우선순위가 결정되었다면, 이익을 제공하기 위해 최고의 잠재력을 가진 테스트 케이스가 먼저 자동화될 것이다. 만일 시간적인 제약이 테스트 자동화의 적용에 영향을 미친다면, 최소한 우리는 최대한의 이익이 도출될 것이라는 점을 이해할 수 있다.

실수는 다음과 같다.

"모든 테스트를 자동화하는 것은 우리에게 너무나 즐거운 일이었어요."

그건 즐거운 일일 수 있다. 하지만, ROI는 명백히 최적은 아니다. 어떤 테스트의 자동화에 소요된 시간은 결코 되돌릴 수 없을 것이다. 일반적인 법칙은 가장 낮은 투자 비용으로 최대화 이익을 내기 위해서 테스트 케이스중 60%가 결코 자동화되어서는 안 된다는 것이다.

베스트 프랙티스 9: 자동화를 개발 프로젝트처럼 관리하라

테스트 자동화 개발은 소프트웨어 개발 프로세스이다. 비록 좀처럼 그런 식으로 다루어지지 않지만 말이다. 소프트웨어 개발 프랙티스를 따르는 것은 자동화된 테스팅 프로젝트의 성공과 실패를 가르게 한다.

우리는 다른 소프트웨어 개발 프로젝트를 수행하듯이 테스트 자동화 프로젝트를 수행해야 한다. 다음의 내용이 테스트 자동화 프로젝트에 따라오게 되며, 이것은 개발 프로젝트에도 적용된다.

프로젝트는 테스트 자동화 개발에 전담 개발자를 필요로 한다.

테스트 자동화는 프로그래머가 해당 테스트의 전문가가 아니므로, 전문 테스터가 조언하거나 요구사항을 전달해야 한다.

테스트 자동화의 이익은 코딩을 시작하기 전에 우리의 접근법을 디자인할 때 나타나게 된다.

테스트 자동화 코드는 추적되고 안전하게 보호되어야 한다. 그러므로, 우리는 소스 코드 관리 시스템의 사용이 필요하다.

테스트 자동화는 버그를 가지게 된다. 그러므로, 우리는 버그를 추적할 계획이 있어야 하며, 그들을 테스트해야 한다.

유저들은 어떻게 사용하는지 알아야 할 것이다. 그러므로, 유저 문서 (user documentation)가 필요하다.

테스트 자동화 동안에 완료된 작업의 유형은 소프트웨어 코드 개발이다. 따라서 이들이 프로그램이므로, 이들도 어플리케이션 코드가 관리되는 것과 같은 방식으로 관리되어야 한다.

실수는 다음과 같다.

"코드가 완료되었을 때, 테스터에게 주어 그들이 테스트 자동화를 구현하도록 하자."

왜 우리가 소프트웨어 개발에 단계별 접근법을 따라야 하나? 테스트 자동화 스크립트 개발은 예외인가? 우리는 똑같이 엄격한 프로세스를 적용해야 하며, 테스트 자동화 스크립터들도 개발자처럼 단계를 따라야 한다. 그리고, 그 스크립트도 테스트되어야 한다.

"당신은 테스트 자동화에 있어서는 SDLC 의 모든 단계를 따를 필요가 없다."

우리가 소프트웨어를 테스트하기 위해 소프트웨어를 작성한다는 사실과 비즈니스 시나리오를 수행하는 것이 아니라고 해서, 방법이 달라야 함을 의미하는 것은 아니다. 하나에 중요한 것이 다른 것에도 중요하다. 둘다 소프트웨어 개발 프로젝트이다.

베스트 프랙티스 10: 올바른 리소스를 사용하라

자동화 테스팅을 열망하는 조직에 의해 지속적으로 간과되는 또 다른 영역은 인력 이슈이다.

자동화된 테스팅은 다른 유형의 테스팅과는 다르다.
대부분의 툴들이 스크립트를 작성하기 위해 절차적인 언어를 사용한다.
이러한 툴들은 IDE (Integrated Development Environments)와 매우 유사하다.
디버깅 스크립트는 전통적인 디버깅 소프트웨어와 같다.

그러므로, 다음과 같은 리소스들이 필요하다.
학술적 수준의 절차절 코딩 언어의 지식
기본적인 코딩 프랙티스와 절차에 대한 지식
테스트 자동화에 사용되는 툴에 대한 심화있는 지식과 실무적인 경험

그러므로, '레코드&플레이백' (대개 백엔드 스크립팅을 배울 필요가 없는)은 대다수의 테스트를 자동화하는데 적합한 방법이 아니다. 이러한 툴들의 사용은 개발 노력과 유사하게 발전한다. 이런 경우에 이것을 실현하기 위해서는 조직 내부에 지원이 있어야 하며, 적절한 인력을 고용해야 한다. 또한 개발에 필요한 시간도 주어져야 한다.

이것을 실행하기 위해서는 테스트 자동화 툴 전문가 또는 비슷한 직책이 만들어져야 한다. 그리고 최소한 한 명의 시니어 레벨의 프로그래머가 배속되어야 한다. 그 프로그래머가 능숙하게 다룰 수 있는 언어가 무엇인지는 정말로 문제가 되지 않는다. 중요한 것은 이 인력이 디자인, 개발, 테스팅, 디버깅 그리고 코드 문서화의 능력이 있어야 한다는 점이다.

테스트 자동화는 테스팅과 개발의 조합이다. 당신의 팀과 리소스를 지도하고 가르치기 위해서, 초기에는 컨설턴트를 활용하라. 이런 컨설턴트들로 부터 가능한 한 많이 배워서, 장애물들을 회피하도록 하고, 그들이 떠난 이후에도 자동화 노력이 성공적으로 진행되도록 하라.

실수는 다음과 같다.

"우리는 테스터를 테스트 자동화 코스에 보냈다. 그런데 그들은 여전히 테스트 자동화 스크립트를 구현하지 못한다."

IDE 에 통합된 테스트 툴을 배우는 것은 훌륭한 스크립터를 만들어내지 못한다. 3 일짜리 코스는 프로그래밍의 원리를 가르치지 못할 것이다. 개발자에 의한 멘토링은 배움의 프로세스를 가속화시킬 수 있다. 테스트 스크립터인 사람을 테스트 팀에 충원하라. 이 인력은 코드를 가지고 작업하는 것에 편안함을 느껴야 하고, 테스트 분석가에 의해 디자인된 기본적인 테스트를 다룰 수 있어야 한다. 그리고 그것을 자동화된 스크립트로 변환할 수 있어야 한다.

"그 테스터는 벤더 교육에 참석하지 않았다."

벤더 교육은 제품에 대해서 당신의 인력을 교육시킬 것이다. 누군가에게 MS 의 개발 도구를 가르친다고 해서 그가 코드 작성자나 프로그래머가 되는 것은 아니다. 그 인력은 단순히 코드를 작성하기 위해 IDE 를 어떻게 사용해야 하는지를 알게 될 것이다. 당신의 인력이 테스트 자동화 담당자가 되기 위해서는 벤더 교육 그 이상이 필요하다. 나는 테스트 팀이 초기의 장애물들을 제거하기 위해서 테스트 자동화 컨설턴트를 고용해 지원을 받을 것을 추천한다. 컨설턴트는 인력을 훈련시키고 멘토링을 하며, 가장 일반적인 함정들을 어떻게 회피할 것인지를 가르칠 수 있다. 간단하고 기본적인 스크립팅 개념으로 부터 시작하고, 이후에 복잡한 것들을 추가하라.

베스트 프랙티스 11: 유지보수를 개발하라

올바른 자동화 아키텍쳐 계획이나 디자인이 없다면, 테스트 팀은 곧 수백, 수천의 테스트 스크립트와 수천개의 개별 실행 결과 파일과 로그 그리고, 다양한 버전의 시스템을 위한 기존 스크립트의 유지보수와 연관된 작업들, 또한 기능 개선을 위한 신규 스크립트의 작성에 직면하게 되는 자신을 보게 될 것이다.

많은 조직들이 겪는 어려운 유지보수 문제를 회피하기 위해서, 다음의 지적이 도움이 될 것이다.

재사용 가능한 아키텍쳐를 작성하라.

범용적이고, 어플리케이션 간에 호환되는 초기화와 설정 파라메터를 개발하라.

재사용 가능하고, 어플리케이션 간에 사용할 수 있는 기능을 작성하라.

컴포넌트의 결합에 의해 테스트 스크립트를 디자인하라.

테스트 자동화 구현에 올바른 계획과 디자인을 통해 한 번 쓰고 버리는 자동화의 작성을 지양하라.

실수는 다음과 같다.

"우리는 유지보수를 염두에 두고 디자인하지 않았다."

만일 당신이 유지보수를 위해 테스트 자동화를 디자인하지 않았다면, 아마도 이미 유지보수의 악몽의 희생자가 되어 있을 것이다. 당신의 현재 스크립트를 리뷰하고, 다른 접근법을 취한다면 (유지보수에 집중해서) 얼마만큼의 스크립트/코드 조각들이 재사용될 수 있는지 판단하라. 만일 재작업의 양이 많다면, 올바른 접근법을 염두에 두고

당신이 들인 노력을 다시 초기화하는 것이 나을지도 모른다.

"스크립트 작성에 어떤 표준도 준수되지 않았다."

이것 또한 유지보수 단계에서 어려움을 겪는 원인이거나 스크립트 작성에 들인 리소스가 허망하게 사라지게 된다. 코딩과 스크립팅의 표준은 전체 팀이 유지보수를 손쉽게 할 수 있게 한다. 서로 다른 사람들이 각자의 스크립트를 쉽게 이해할 수 있으며, 누군가 그 팀을 떠나는 경우에도 세상이 무너져버리는 어려움은 없다. 테스트 자동화 담당자가 잠시동안 작업을 중단했던 스크립트로 돌아와서도 그들이 맨처음에 무엇을 작업했었는지 쉽게 이해할 수 있다. 이것은 스크립트의 유지보수 시간을 감소시킨다.

당신의 프로젝트/조직내의 테스트 자동화 담당자나 스크립터가 좋은 코딩 규칙과 표준을 가지고 구현하도록 하라. 이러한 프랙티스는 언제 시작해도 늦지 않다.

베스트 프랙티스 12: 각각의 프로젝트를 마치고 난 후 리뷰와 개선 프로세스를 시행하라

개발 프랙티스는 프로젝트의 종료시점에 '포스트 모텀'을 가질 것을 제안한다. 이것은 테스트 자동화 프로젝트에서도 여전히 유효하다. 다음의 정보가 '포스트 모텀' 동안에 드러날 것이다.

그 프로젝트 동안에 발견되었던 특정 이슈를 다루는 새로운 방식이 아마도 제안될 것이다.

축적된 경험이 현재 프로세스나 사용하고 있는 아키텍쳐에 대해 요구되는 변경에 대한 통찰력을 제공할 것이다.

개선과 현재 프로세스가 좀더 효율적이 될 수 있는 아이디어의 도출

리뷰되어야 하는 메트릭이 프로세스의 개선을 위한 가치있는 정보를 제공할 것이다.

더 많은 훈련이나 멘토링의 필요성이 나타날 수 있다.

당신의 프로세스를 개정하고 배우기 위해서 '포스트 모텀으로 부터 그리고, 테스트 자동화 구현물에서 최대회 이익을 얻기 위해 베스트 프랙티스에서 나온 정보를 사용하라.

실수는 다음과 같다.

"우리는 우리가 첫번째로 조직에서 테스트 자동화 프로젝트를 했던 것과 동일한 방식으로 여전히 테스트 자동화를 구현하고 있다."

팀/조직이 경험으로 부터 배우지 않았다는 것이 명확하다. 우리의 경험에서 배우는 것은 우리가 더 나은 방식으로의 변화, 최대 이익을 위해 프로세스를 개선, 우리 자신을 위해 삶을 더 용이하게 만들어 준다. 그 테스트 팀은 어떤 일이 효과가 있었고, 어떤 것이 진행되지 않았는지 리뷰할 필요가 있으며, 장래의 프로젝트를 위해 자신들의 방식을 개정해야 한다.

"이러한 자동화의 노력을 통해 우리가 얻은 이익은 무엇인가?"

이것은 우리가 베스트 프랙티스 1: 목적을 분명히 하라를 충실히 따랐다면 쉽게 답할 수 있어야 한다. 우리는 테스트 자동화를 구현할 때 목표를 염두에 두고 시작했다. 그리고, 우리는 우리의 목표를 향해서 작업했다. 프로젝트가 완료되고 나서, 우리는 계획했던 것을 달성했는지 알아야 한다. 만일 우리가 답하지 못한다면, 베스트 프랙티스 1 번을 실행할 때이다.

결론

당신 회사의 프로파일을 판단한 이후에, 당신의 프로세스를 완전하게 하고, 테스트 전문가들을 양성하고, 팀 구성원에게 적절한 테스팅 툴을 제공하고, 이 아티클에 나와있는 베스트 프랙티스들을 따르면, 당신의 회사는 자동화된 소프트웨어 테스팅의 이익을 실현하게 된다. 수동 테스팅과 비교하여 적절하게 적용된 자동화는 높은 품질의 제품, 회사에 낮은 리스크, 빠른 승인 그리고 시장에 출시 시간을 감소시키는 결과를 가져올 수 있다.

자동화된 소프트웨어 테스팅의 성숙도의 사다리에서 당신이 더 높은 수준에 도달한다면, 더 많은 이익이 실현될 것이다. 당신이 선택한 수준이 어떤 것이든 간에, 지난 수십년에 걸친 다수의 교훈을 마음에 새겨라. 당신이 구매한 툴이 어떤 것이든 간에, 지금까지 당신이 가장 크게 투자한 곳은 이러한 툴들을 사용하기에 적절한 인력과 프로세스일 것이다.

성공적인 테스트 자동화 구현에 이르도록 안내자가 이끄는 것처럼 베스트 프랙티스트를 사용하고, 당신의 테스트 자동화 구현에서 배움으로써 조직의 개별적인 필요에 맞게 이러한 프랙티스들을 적용하라.

Henk Coetzee

참고:

1. http://web.archive.org/web/20040304002324/wwww.globaltester.com/sp9/removing.html

{끝}