성공적인 테스트 자동화를 위한 방법

- 테스트 매니저가 알고 있어야 할 10가지 함정들

Matthias Daigl

요약

테스트 자동화는 매우 복합적인 작업이며 몇 가지 이슈들이 테스트 자동화에 걸림돌이 될 수 있습니다. 성급하게 테스트 효율을 높이려고 하는 일은 실패하기 매우 쉽습니다. 이 문서에서는 테스트 자동화 프로젝트 전 과정에서 나타날 수 있는 일반적인 문제들을 기술해 보고 10가지 성공 사례들을 알아 보고자 합니다.

문서 키워드 : Software test automation, best practice

1. 소개

오늘날의 소프트웨어 산업에서 수동 테스팅 과정은 노동 집약적이며, 극단적으로 시간과 비용을 소모하며 때론 단조롭고 소프트웨어 엔지니어와 테스터들에게 있어 그리 환영 받지 못하는 업무입니다.
가능성이 있는 해결책으로는 상업적으로 이용가능한 컴퓨터 기반의 테스팅 툴이 제시될 수 있습니다. 일반적으로 성공적인 자동화 사례는 테스트 자동화에 적합한 케이스들 입니다.
의미를 명확하게 구분하기 위해서 이 문서에서는 GUI Blackbox-Testing에 대해서만 논의하고자 합니다.

첫째로 다음과 같은 GUI-testing tool(Capture-and-Replay tools)이 있을 수 있습니다.

상업적으로 이용 가능한 대부분의 Capture-and-Replay tools(CR-Tools)은 거의 모두 비슷한 기능과 동작을 가지고 있습니다.

Capture mode : 모든 CR-Tool은 테스트 과정중에 테스트 대상과 사용자가 주고 받는 상호 작용을 모두 기록합니다. 모든 CR-Tool은 객체에 기반하고 있습니다. 말하자면, 사용자가 버튼이나 라디오 박스, 툴바와 같은 GUI 요소를 선택하는 행동을 인식합니다. 그리고, XY 좌표계에 기반한 마우스 클릭이 아닌 모든 인식 가능한 객체의 특성들(이름, 색상, 레이블, 값)을 기록합니다.

Programming : 기록된 모든 요소들은 CR-Tool에 의해 C 언어나 Basic 언어와 유사한 형태의

테스트 스크립트로 저장됩니다. 또한, 프로그래밍 언어의 모든 기능들(문자 인식, 루핑, 서브루틴)을 가지고 있으며 더 복잡한 테스트 케이스들을 수행할 수 있습니다.

Checkpoints : 프로그램이 테스트 되고 있는 중인지, 기능은 정상 동작하고 있는지, 에러가 발생했는지를 확인하기 위해서 (테스트 캡쳐 과정중이나 스크립트를 수정하는 과정에서) 부가적인 체크포인트를 스크립트 안에 추가 할 수 있습니다. 이러한 일련의 방법으로 윈도우 객체의 UI와 관련한 특성들(색상, 위치, 크기 등)이나 테스트 대상의 기능적 특성들(값, 메시지 박스의 내용 등)을 확인할 수 있습니다.

Replay mode : 한 번 캡쳐된 이후에 테스트는 언제라도 다시 수행될 수 있습니다. 앞에서 얘기한 바와 같이 객체에 기반한 테스트 캡쳐는 테스트가 수행될 때마다 다시 GUI 요소들을 새로 인식합니다. 이런 동작은 소프트웨어의 수정에 의한 GUI의 변경이 발생한 이후에도 계속됩니다. 만약 테스트 프로그램이 이전에 캡쳐된 테스트가 다시 수행될 때와 다르게 동작한다면, 체크 포인트에서 체크되어 테스트는 실패하게 됩니다.

이러한 경우 CR-Tool은 테스트 프로그램에서 에러가 발생했다고 기록하게 됩니다.

상업적인 CR-Tool을 제작하는 업체들은 자신들이 판매하는 툴이 가장 빠르며 테스트 자동화에 가장 쉬운 방법을 제공한다고 선전합니다. 그러나, 현실적으로 CR-Tool에 근거한 테스트의 효율성을 해칠 수 있는 많은 함정이 존재합니다.

이러한 함정들은 테스트 자동화 프로젝트에서 매우 다양한 형태로 발생할 수 있으며 각각의 주제들을 좀 더 자세히 살펴보도록 할 것입니다. 아래에서 10가지의 주제에 관해서 이야기를 계속 해 봅시다.

2. 함정의 유형들

테스트 자동화는 실패하거나 성공할 수 있습니다. 일반적인 함정들을 피해가는 방법이 있긴 합니다. 문제와 그에 대한 해결책은 우리가 함정 유형이라고 부르는 것에 있습니다. 물론, 이보다 훨씬 더 많을 수는 있습니다만 가장 일반적이고 중요하다고 생각되는 10가지 함정 유형을 아래와 같이 구별할 수 있습니다.

- 기적을 믿음(너무나 많은 기대를 가지고 있음)
- 너무 이른 시점의 자동화와 너무 늦은 시점의 자동화(자동화 시점의 문제)
- 이른바 Iterative 소프트웨어 개발 방법론(변경 또는 추가된 요구 사항을 반영하는 시점의 문제)
- 테스트 자동화를 해야 유리한 부분과 그렇지 않은 부분(통제력을 상실하는 경우)

- 테스트 용이성이 충분한 개발 방법
- 자동화 테스트의 단위 모듈화와 테스트 스크립트 재사용성
- Regression Test의 함정
- 테스트 환경(OS, 시스템, 하드웨어 등등...)
- 자동화 테스트를 가능하게 하는 기반 들(문서, 시스템, 소프트웨어, 절차 등등...)
- 테스트 파트와 개발 파트가 분리 되어 있음(위치적, 조직적으로...)

이제 부터 각각의 항목에 대해서 좀 더 자세히 알아 봅시다.

2.1 기적을 믿음

책임 범위 : QA 관리자, 일반 관리자
프로젝트 단계 : 자동화 툴의 선택, 소개

시나리오 :

시간과 비용이 많이 드는 테스트 과정은 자연스럽게 자동화 테스트를 추구하게 됩니다. "우리는 툴이 있어야 해!"
상용 자동화 툴 업체의 영업 활동과 제품의 높은 가격은 그것이 가져올 효과에 대해서 기대를 하게 합니다. 제품을 시험 평가하는 과정에서 몇 가지의 툴이 상용 업체에 의해서 소개됩니다. 회사 내의 전문가 집단이 업체의 제품 소개에 대한 그들의 의견을 내놓고, 가격이나 기능 비교표가 검토 대상이 됩니다.

그 중 하나의 상용툴을 도입한 이후에 그것이 우리에게 아주 적합한 것으로 보입니다.

나중에 어떤 사람이 그 상용툴을 이용해 실제 자동화 업무에 이용했습니다. 처음에는 매우 의욕적으로 자동화 업무를 구현해 봅니다. 그러나, 현재의 테스트 환경에 불리한 점이 있기 때문에 자동화 테스트를위한 툴의 캡쳐링 작업이 잘 동작하지 않습니다. 얼마 지나지 않아, 테스트 작업이 완료되고 Regression 테스트가 계획되었습니다. 하지만, 대부분의 작업은 Regression 테스트를 하기 위해 기존 자동화 테스트를 유지 보수하는데 많은 시간이 허비되었습니다. 이제 그 누구도 테스트 자동화에 관심을 가질 만한 시간이 없을 무렵, 사내에 도입된 상용툴은 쓸모 없는 것으로 간주되어 더 이상 사용되지 않았습니다.

분석 :

대단한 수준의 기대 성과를 내려면 그만한 수준의 노력이 필요합니다. 상용 자동화 툴의 도입을 결정한 사람들은 테스트 자동화에 대해서 그다지 충분한 경험이 없으며 업체의 말을

그대로 믿었습니다. 위의 예에서는 툴의 도입을 위한 선택이 올바르지 않았습니다.

테스트 대상이 되는 어플리케이션과 툴이 적합한지에 대해서 평가하는 과정은 매우 중요합니다. 대부분의 툴에 대해서 평가를 해볼 필요는 없지만, 3-4가지의 유망한 툴에 대해서는 반드시 평가를 시도해 보아야 합니다.

대개 테스트 자동화를 구현하는 업무는 중요하지 않게 생각되기 마련입니다. 하지만, 그러한 과정을 거치지 않고는 아무것도 할 수 없습니다. 시작 하기는 아주 쉬운 것으로 보일지 몰라도 테스트 자동화는 경험이 필요한 대단히 복잡한 작업입니다. 가끔은 실수라고 판단되기까지 매우 오랜 시간이 허비될 수도 있습니다.

해결책 :

평가는 해당 환경에 대해서 이루어져야 합니다. 요구 사항이 사전에 정의되어야 하며, 가장 적절하다고 여겨지는 툴을 선택해야 합니다. 심지어 특정한 환경에서 최적의 솔루션으로 여겨지는 것도 부족한 부분이 있게 마련입니다.

새로운 업무 프로세스와 툴을 도입하는 것은 대단히 막중한 업무입니다. 자동화 툴의 도입으로 인한 예상 결과(기대)는 현실적이어야 합니다. 기대한 만큼의 성과는 쉽사리 얻어지지 않습니다.

툴을 사용하려고 하는 사람 또는 사람들은 그것의 사용법을 배우는데 충분한 시간을 투자해야 합니다. 훈련도 필요하지만, 컨설팅의 조언도 필요합니다.

2.2 자동화 시점의 문제

책임 범위 : 테스트 관리자, 프로젝트 관리자
프로젝트 단계 : 테스트 계획

시나리오 :

자동화 테스트의 개발은 소프트웨어의 프로토타입이 완성되었을 때 가능한 한 빨리 시작됩니다. 테스트 스크립트는 GUI 요소에 기반해서 동작합니다. 테스트 되고 있는 소프트웨어의 구조가 변경될 때마다, 이미 구현되었던 테스트 자동화는 동작하지 않을 뿐더러 유지 보수(수정)를 필요로 합니다. 테스트 자동화를 다시 동작시키기 위한 실질적인 작업은 완료되었지만 그 작업은 처음에는 계획되지 않았던 것입니다.

분석 :

가능한 한 빨리 테스트 케이스가 수행되는 것이 바람직합니다. 그렇게 되면, 테스트를 더 빨리 수행할 수 있고, 버그를 빨리 찾아낼 가능성을 높여서 시간과 비용을 절약할 수 있기 때문입니다.

하지만, 테스트 중인 대상의 디자인이나 UI 요소의 안정성을 먼저 고려해야 합니다. 디자인이나 UI 요소가 안정화 되지 않은 상태라면 테스트 자동화를 달성하기 위해서 대단히 많은 양의 재작업 요소가 발생할 수 있음을 고려해야 합니다.

다른 측면으로는 테스트 자동화가 매우 늦게 시작된다면 그만큼 테스트가 수행되는 것이 더딜수 밖에 없다는 점을 생각해야 합니다.

해결책 :

테스트 작업은 온갖 난관에 대처할 수 있어야 합니다. 소프트웨어의 변경이 없을 수는 없습니다. 테스트 자동화는 가능한 한 빨리 시작해야 하며, 필요하다면 가능한 한 늦게 시작되기도 해야 합니다. 하지만, 테스트 자동화가 시작되었다면 소프트웨어의 주요한 디자인은 이미 확정되어 있는 상태여야 합니다. 특히, GUI 요소는 이미 안정화 단계(더 이상의 수정이 없으며 기능적으로 완전한 상태)에 접어들어야 합니다.

또한, 피할 수 없는 변경이 발생할 때를 대비해서, 테스트 스크립트는 모듈화가 되어야 합니다. 스크립트의 디자인은 유지 보수가 용이하도록 미리 설계되어야 합니다.

2.3 이른바 Iterative 소프트웨어 개발 방법론

책임 범위 : 품질 관리자, 프로젝트 관리자
프로젝트 단계 : 전과정

시나리오 :

Iterative 소프트웨어 개발 방법론은 회사 내부에서 정의 되었습니다. 이 방법론은 소프트웨어 개발에 몇가지 단계를 거치는 것을 반복하는 것입니다. 각 반복 단계마다 기능들이 정의됩니다. 그것은 스펙 디자인, 개발, 테스트 단계들 입니다. 각 단계를 거친 이후에 조금씩 향상된 품질의 제품이 릴리즈 준비를 합니다. 이것이 Iterative 소프트웨어 개발 방법론의 프로세스 모델입니다.

각 반복 단계의 초기에 기능들이 정의될 때, 테스트 자동화 역시 함께 시작합니다. 추가적인 기능에 대한 추가적인 테스트 역시 수행됩니다.

마케팅 적인 고려로 인한 개발 완료의 압력으로 인해(또는 다른 요인 예를 들면, 기능을 전부다 구현할 여력이 안된다던지) 기능들은 한 주기의 개발 과정이 진행되는 동안 변경됩니다. 그럴수록 테스트 자동화 과정은 계속 유지 보수 되어야 하며 꽤 많은 양의 노력을 필요로 합니다.

분석 :

이 경우 Iterative 소프트웨어 개발 방법론을 잘못 이해하고 있는 것입니다. 원래 구현하려고 계획했던 것이 요구에 의해서 다음 번 릴리즈에서(주기가 아닌) 변경되어서는 안됩니다. 만일에 요구 사항이 변경된다면, 그것을 바로 구현하지 말고 다음번 주기까지 미뤄야 합니다.

해결책 :

요구 사항과 기능들은 한 개발 주기 동안에는 변경되면 안됩니다. 유연성있는 작업을 위해서는 cycle-time(한 주기 동안 추가되거나 변경되는 기능의 양)이 짧아야 합니다. 테스트 용이성을 높이기 위해서는 새로운 기능이 추가되더라도 개발팀이 그 나머지 부분에 대해서 변경하지 않아야 합니다.

2.4 테스트 자동화를 해야 유리한 부분과 그렇지 않은 부분(통제력을 상실하는 경우)

책임 범위 : 프로젝트 관리자
프로젝트 단계 : 전과정

시나리오 :

테스트 스펙 문서가 작성되고 테스트 자동화 툴이 준비되면 테스트 자동화를 시작하자고 결정을 합니다. 테스트가 수행될 때 환경적인 요인으로 몇가지 문제가 발생하고 그에 따른 유지 보수 노력이 필요해 집니다. 어떤 테스트 항목들은 구현에 어려움이 있지만 구현이 가능하다면 결국은 구현됩니다. Regression Test에서 Hang이 발생하는 경향이 있는 테스트를 거치면서 테스트는 조금씩 향상됩니다.

테스트 자동화는 수동 테스트에 비해서 훨씬 더 많은 노력이 필요한 방법입니다.

분석 :

자동화 비용에 대한 이해, 비용 회수의 가능성에 대해서는 인정하지만, 자동화 노력에 대한

관리는 자주 무시됩니다. 그럼에도 불구하고, 자동화의 비용에 대해서 이해하고 있다면, 자동화의 일부 노력만으로도 결국 본전은 한다는 사실을 알게 됩니다.

해결책 :

모든 테스트 영역이 자동화하기 적합한 것은 아닙니다. 테스트 자동화는 모든 테스트를 자동화 하라는 것을 의미하지는 않습니다. 어떤 테스트 영역을 자동화 할 것이며 어떤 영역을 수동 테스트 할 것인가 결정하는 것은 매우 중요한 일입니다.

만약에 특정한 테스트 영역이 자주 반복되며 테스트를 수행하는데 매우 많은 노력이 필요한 경우라면 자동화되어야 하며 제품이 릴리즈 된 이후에 수행되는 테스트나, 다양한 플랫폼(OS)에서 또는 다수의 테스트 데이터가 필요한 테스트의 경우도 마찬가지 입니다.

단, 이 경우 안정적인 GUI가 반드시 요구됩니다.

테스트 자동화와 그 자동화를 지속하기 위한 유지 보수는 테스트를 수행하는데 들어가는 비용 대비 효과를 잘 따져서 수행되어야 합니다.

2.5 테스트 용이성이 충분한 개발 방법

책임 범위 : 개발팀
프로젝트 단계 : 개발

시나리오 :

결과적으로 볼 때 소프트웨어는 테스트 작업을 수행하기 위해서 제작되는 것으로 볼 수 있습니다. 일반적으로 Capture and replay는 잘 동작하지만, 소프트웨어의 특정한 요소들은 테스트 하기에 무척 복잡합니다. 예를 들면, 각기 다른 윈도우의 레이블은 거의 동일하며(구별하기 어렵게) 결과를 수집하기도 어렵고 Custom Control들은 화면 좌표계에 기반한 마우스 클릭으로 밖에는 조작할 수 없으며 레이블은 쉽게 변하고 윈도우의 내용도 자주 변경됩니다.

따라서, 이러한 요소들의 테스트 수행 과정은 다른 것에 비해서 매우 많은 노력을 필요로 하며, 마찬가지로 유지 보수를 위한 노력도 증가합니다.

분석 :

테스트 팀은 테스트에 필요한 툴과 방법을 선택하지만, 테스트 대상은 선택할 수 없습니다.

사실 테스트 용이성은 좀처럼 제품 개발(디자인) 과정에 고려되지 않습니다. 개발팀이 테스트 팀에게 던져주는 문제에 대해서 알지 못하기 때문에, 대부분의 소프트웨어를 테스트하는 과정은 매우 어렵습니다.

일반적으로 테스트 용이성을 증가시키는 방법은 구현하기가 쉽습니다.

해결책 :

만일 일찍 개발팀에게 그러한 테스트 용이성을 증가 시킬 수 있는 방법에 대해서 언급한다면 테스트 용이성의 정도에 영향을 미칠 수 있습니다. 초기 버전이 이미 릴리즈 된 이후에 구조를 변경해야 하는 요청 사항을 전달하는 것은 적절하지 않습니다. 테스트 팀은 개발 초기 단계 이전 부터 테스트 자동화에 적합하지 않은 요소를 분석하고 그러한 것을 사전에 막을 수 있는 이슈들을 정리해서 개발팀에 전달해야 합니다. 이러한 방법은 프로젝트에 불필요한 장애들을 피할 수 있는 방법입니다.

2.6 자동화 테스트의 단위 모듈화와 테스트 스크립트 재사용성

책임 범위 : 테스트 엔지니어
프로젝트 단계 : 테스트 수행

시나리오 :

테스트 팀은 CRM 솔루션의 테스트 작업을 수행하고 있습니다. 테스트에는 새로운 고객을 추가하거나, 리스트를 생성하고, 고객을 소트하거나 레코드를 삭제하는 과정이 포함되어 있습니다.

어떤 테스트는 다른 테스트 작업에 의존적이도록 만들어 졌습니다. 예를 들면, 고객 리스트를 만들거나 소트하는 과정은 고객의 추가가 먼저 선행되어 있어야 합니다.

테스트 과정이 처음 수행되었을 때 모든 것이 잘 작동하는 것처럼 보였습니다. 그러나 몇 주 후에 Regression Test가 계획되고, 개개의 테스트 과정이 중요도를 가지고 있기 때문에 모든 테스트 스크립트를 전부 수행할 만한 시간이 없습니다. 초기에 중요한 사전 조건이 되는 기능에 대한 테스트가 적절히 수정되지 않았기 때문에 어떤 테스트 들은 동작하는데 필요한 조건이 만족되지 못해서 실패하게 됩니다.

분석 :

각기 다른 작업에 각각 독립적인 테스트 스크립트를 제작하는 것은 매우 좋은 생각입니다. 하지만, 시나리오에 있어서 이미 정의되어 버린 테스트는 독립적이지 않습니다. 그 테스트 들은 테스트가 수행되기 위한 사전 조건을 체크하지 않으며, 최소한의 동작 조건을 충족하는지도 확인하지 않습니다.

별다른 중요도가 존재하지 않는다면, 어떤 자동화 테스트 모듈이던 간에 순서없이 수행될 수 있어야 합니다.

해결책 :

테스트 과정은 모듈화가 되어 있어야 합니다. 각각의 독립적인 테스트의 사전 조건과 사후 조건은 정확하게 정의되어 있어야 합니다.

일반적으로 자동화 테스트 스크립트는 그 첫 부분에 동작 가능했던 시점과 동일한 상태로 초기화 하는 과정을 가지고 있어야 합니다. 각 테스트 스크립트는 자신이 수행되는데 필요한 조건이 만족되는 상태인지 확인해야 합니다. 그렇지 않다면, 그 자신이 그 조건을 만들어야 합니다.(예를 들면, 다른 스크립트를 호출한다던지)

이것이 가능하지 않다면 그에 해당하는 메시지를 로그에 기록하고 스크립트는 중지되어야 합니다.

2.7 Regression Test의 함정

책임 범위 : 테스트 엔지니어
프로젝트 단계 : 테스트 수행

시나리오 :

테스트 자동화가 소개되고 난 후에 첫번째 테스트 자동화가 Capture and replay 방법으로 생성됩니다. 테스트 자동화 작업은 다시 한 번 수행해도 매우 잘 동작하게 됩니다. Regression Test가 두 번 수행되었습니다. 테스트 과정중에 있던 어플리케이션이 수정되었지만, 버그가 수정되는 정도였으며 새로운 기능 추가는 없었습니다. 테스트 팀은 매우 만족해 했으며, 매뉴얼 테스팅 방법과 비교했을때 매우 쉽게 만족스럽 결과에 도달했습니다.

다음번 Regression Test 과정에서 새로운 요구 사항이 추가되었고 소프트웨어의 인터페이스가 변경되었습니다. 몇몇의 테스트 스크립트가 더 이상 동작하지 않게 되었습니다. 다양한 스크립트의 유지 보수가 이어 졌습니다. 그 시점에서는 더 이상 테스트 자동화는 비용 절감의

방법이 아닌 것처럼 보이게 되었습니다.

분석 :

Regresstion Test에는 두 가지 방법이 있습니다. 단순히 버그가 수정되는 경우와 어플리케이션의 기능이 늘어나는 경우입니다. 첫번째 경우에는 테스트를 다시 수행하는 것은 매우 쉬운 일입니다. 두 번째의 상황은 좀 더 복잡합니다. 새로운 기능의 추가 때문에 어플리케이션을 조작하는데 영향이 있습니다.

테스트 스크립트의 잠재적인 문제가 단 시간내에 밝혀지지는 않습니다. 하지만, 테스트 중인 소프트웨어의 변경은 불가피 하게 발생합니다.
좋은 테스트 스크립트는 이러한 상황에 대해서 대처가 가능해야 합니다.

해결책 :

Capture and replay 방법은 테스트 자동화를 구현하는 한가지 방법입니다. 구현하기 쉽기도 하지만, 프로그래밍을 통해서 캡쳐된 테스트의 수정을 가할 수 있다는 점이 최고의 장점입니다. 이것은 테스트 엔지니어 들의 경험을 통해 잘 알려진 사실입니다. 테스트 자동화의 초심자들은 빠른 결과를 알 수 있고, 매우 간단한 솔루션을 선호하는 경향이 있습니다.

테스트가 자동화 되려면 소프트웨어에는 잠재적인 변경이 항상 발생할 수 있다는 점을 고려해야 합니다. 스크립트의 유지 보수는 때를 가리지 않고 어느 때나 필요하게 됩니다. 그러한 이유로, 스크립트는 유지 보수가 쉬운 방법을 가지고 있어야 합니다.

테스트 자동화의 초기 단계에는 많은 노력이 필요하지만, 자동화가 계속 진행될 수록 그 노력은 이전보다 적게 들수 있습니다.

2.8 테스트 환경

책임 범위 : 테스트 엔지니어
프로젝트 단계 : 테스트 수행

시나리오 :

테스트가 매우 잘 수행되고 있습니다. 얼마 후에, 테스트를 다시 한 번 수행할 계획을 세웠습니다. 놀랍게도, 아무런 문제가 없는데 테스트가 수행되지 않습니다. 만약, 그렇다면 테스트 스크립트는 정해진 테스트 데이터에만 의존해서 결과를 리포트 하고 있는 것입니다.

예를 들면, 데이터가 더 이상 입력되지 않는다면 이 시스템에는 이미 데이터가 입력되어 있는 것입니다.

문제의 원인이 명백하게 테스트 데이터에 있는 것이라면, 시스템은 다시 초기화 되어야 합니다. 최대한 안전하게 작업이 되려면, 운영체제 또는 서비스 팩, 어플리케이션 모두 다시 설치되어야 합니다. 데이터 베이스는 다시 기본값으로 초기화 되어야 합니다. 그럼에도 불구하고, 테스트가 동작하지 않습니다. 아마도, 서비스 팩의 버전이 틀리게 설치되었거나 인터넷 익스플로러의 버전이 틀린 것 같습니다.

분석 :

문제는 자동화 테스트 때문이 아니라 테스트 환경에 있습니다. 테스트가 수행될 때마다 모든 조건들이 이전 테스트가 완료되었을 때와 동일해야 합니다. 상이한 운영 체제와 같이 사전 조건이 변경되는 것이 테스트의 일부인 경우는 예외입니다.

하지만, 일반적으로 테스트 환경은 100% 똑같이 구현이 가능합니다. 테스트 환경은 테스트 대상으로만 구성되는 것이 아닌, 운영체제, 서비스 팩, 패치, 데이터 베이스 등등의 요소가 모두 포함됩니다.

해결책 :

이미지를 생성하는 방법은 100% 동일한 테스트 환경을 제공할 수 있는 하나의 방법입니다. 하지만, 어떤 이미징 툴을 사용하느냐를 결정하는 것만이 고려되어서는 안됩니다. 각기 다른 이미지들의 내용에 대해 기술해 놓은 정의 문서가 반드시 필요합니다. 문서와 이미지는 따로 존재해서는 안됩니다. 각자 한가지 씩만 존재한다면 그들은 쓸모없게 됩니다.

2.9 자동화 테스트를 가능하게 하는 기반 들

책임 범위 : 테스트 엔지니어
프로젝트 단계 : 전과정

시나리오 :

우리 회사내에서 이루어지는 소프트웨어 패키지의 모든 테스트 과정은 한 명의 경험많은 테스트 엔지니어에 의해서 이루어집니다. 그는 3년 동안 그 일을 해왔습니다. 테스트 스크립트들은 아주 잘 동작하며, 테스트의 자동화 정도는 85% 정도로 매우 훌륭한 수준입니다. 어느날 그는 회사를 그만 두었고 그 자리는 다른 사람이 맡게 되었습니다.

운좋게도 후임자 역시 그 일에 매우 적임자 였습니다. 그녀는 테스트 자동화와 테스트 중인 시스템에 대해서 매우 잘 알고 있었습니다. 그녀의 첫번째 업무는 고객이 옛날 버전의 제품에 대해서 문제를 제기한 데 대해서 테스트하는 것이었습니다. 옛날 버전의 제품은 버전 관리 시스템으로 인해서 잘 설치가 되었지만 테스트 스크립트는 동작하지 않았습니다. 결국, 테스트 스크립트에 대한 문서가 부족하다는 문제가 발견되었고 테스트 스크립트 자체가 매우 효율적으로 작성되어서 이해하기가 매우 어렵다는 사실을 이해하게 되었습니다.

분석 :

테스트 스크립트는 잘 관리되지 않았습니다. 아무도 테스트 스크립트는 소프트웨어라고 생각하지 않았습니다. 따라서, 테스트 스크립트와 관련한 문서는 존재하지 않았습니다. 테스트 스크립트는 버전 별로 관리되어 있지 않았습니다. 테스트 스크립트는 매우 효율적으로 작성되어 있어서, 원 저작자만 유지 보수가 가능하도록 작성되어 있다는 사실을 알 수 있었습니다.

해결책 :

테스트 자동화 과정의 결과는 그 자체로 테스트 기반입니다. 바로 실행 가능한 테스트 스크립트는 그 값어치 이상으로 생각될 때가 있지만, 사실은 테스트 스크립트를 생성해 내는 기술 자체가 매우 값어치 있는 것이며 그 자체로도 소프트웨어라고 볼 수 있습니다. 테스트 시스템의 복잡성은 종종 소프트웨어 그것과 비교될 때가 있습니다.

만약 테스트 기반(자동화 테스트를 가능하게 하는 Infra)이 소프트웨어나 툴과 같다고 생각된다면 그것 역시 동일한 프로세스와 툴로서 취급되어야 합니다. 스펙 작성, 리뷰, 리허설, 코드 검사, configuration management, change management 등의 모든 일반적인 소프트웨어 프로젝트에서 사용하고 있는 품질 관리 기법들이 테스트 기반에도 그대로 적용되어야 합니다.

2.10 테스트 파트와 개발 파트가 분리 되어 있음

책임 범위 : 프로젝트 관리자
프로젝트 단계 : 초기

시나리오 :

우리 회사는 테스트 파트와 개발 파트가 떨어져 있습니다. 테스트 자동화와 개발 모두 적절한 역량을 가진 인원들에 의해서 이루어 졌습니다. 두 팀 모두 요구 스펙에 맞게 그들의 작업을

진행했습니다.

테스트가 시작되었을 때, 개발파트와 테스트 파트가 몇몇 요구 사항을 각기 다르게 이해하고 있음이 밝혀졌습니다. 많은 수의 버그 리포트가 작성되었지만, 대부분은 개발 파트에 의해서 버그가 아닌 것으로 판명되었습니다. 테스트가 수행되고 난 이후에야 요구 사항에 대해서 각 주체가 모두 적절하게 이해하고 있는지가 분명해 졌습니다.

분석 :

개발 파트와 테스트 파트가 분리되어 있는 경우 더 많고 정밀한 의사 소통이 요구됩니다. 하지만, 업무가 좀 더 명확해 진다면 명확해 지지 않았을 때보다는 의사 소통에 덜 시간을 소비하게 됩니다.

각 개발 주체는 회사의 다른 부서로 부터 승인을 받지 않은 스펙들(테스트 스펙, 디자인 스펙)을 빼고는 그들이 의도하고 있는 바를 상세히 기술해야 합니다.

분석 :

일반적으로 테스트 파트를 개발팀과 분리하는 것은 좋은 생각입니다. 기본적으로 테스트 업무와 개발 업무는 매우 다른 자격 요건을 필요로 합니다. 실제 개발 과정으로 부터 멀어지면 멀어질 수록 테스트는 좀 더 객관적으로 작성될 수 있습니다.

각각의 그룹 간의 의사 소통은 계속 향상되어야 합니다. 스펙 문서는 하나의 그룹이 아닌 각기 다른 부서의 대표들에 의해서 검토되어야 합니다. 기본 스펙을 이해하고 있는 정도가 동일한지에 대한 내용 뿐만 아니라 그 이상의 것도 확인되어야 합니다. 테스트 파트는 이러한 과정을 통해서 더 쉽게 테스트 할 수 있는 방법(비용이 적게 드는)을 개발 팀에 요구할 수 있습니다.

개발팀은 테스트 스펙 문서를 읽음으로서 테스트가 수행되기 전에 잘못된 점을 지적하거나 테스트가 수행되면 어느 정도의 버그가 발견될 것인지에 알 수 있게 됩니다.

3. 결론

지금까지 언급된 내용은 다양한 소프트웨어 자동화 프로젝트의 컨설팅을 수행하면서 얻은 경험을 정리한 것입니다. The experiences of the imbus testing laboratory have complemented these scenarios, and some of the solutions developed in the laboratory – processes, communication channels, tools – have been introduced to prevent those patterns.
Most problems occur because testing and test automation are merely regarded as a part of

quality management, but not as a complex process. It demands the methods of quality management to be applied.
Learning from own failures may be most impressive, but we should try to learn from others' experiences as well. The pitfall patterns presented in this paper may guide you on your road to successful test automation.

Literature

[1] Common Mistakes in Test Automation, Mark Fewster, Grove Consultants, 2001, presented at ICS Test, 4.4.-6.4.01, Bonn
[2] Classic Testing Mistakes, Brian Marick, http://www.testing.com/writings/classic/mistakes.html
[3] Success with Test Automation, Bret Pettichord, BMC Software, 1996, http://www.io.com/~wazmo/succpap.htm
[4] Automated Testing of Graphical User Interfaces, Results of the ESSI PIE 24306, Tilo Linz, Matthias Daigl, imbus GmbH 1998, in AQuIS '98, Proceedings:"4th International Conference on Achieving Quality In Software", 30.3. - 2.4.98, Venice.

Author

Matthias Daigl
imbus AG
Kleinseebacher Strasse 9
D 91096 Moehrendorf, Germany
Tel: 09131/7518-0
Fax: 09131/7518-50
Email: matthias.daigl@imbus.de

Matthias Daigl was born in Hamburg 1968. He received his diploma in information science from the Friedrich-Alexander-Universitat Erlangen/Nurnberg in 1995. After working as a software engineer he joined imbus in 1997. As part of the "ESSI"-Initiative, Matthias was selected to be Project Test Manager of the PIE-Project "Automated Testing of Graphical User Interfaces". Since then he has been involved in test automation. As a project leader, consultant and trainer he has gained comprehensive insight in both technical and organizational issues of test automation.