Exploratory Testing Explained By James Bach

탐색적[1] 소프트웨어 테스팅은 강력한 테스팅 기법 중의 하나이지만, 광범위하게 잘못 이해되고 있는 면이 있다. 어떤 상황에서는, 탐색적 테스팅이 스크립트 테스팅 보다도 더 높은 생산성을 가질 수 있다. 아무리 간단한 테스트를 수행한다고 하더라도, 모든 테스터들은 어떤 형태로라도 탐색적 테스팅을 수행하고 있다. 몇몇 사람들이 이 기법에 대해 연구한 적이 있지만, 소프트웨어 테스팅 분야에서 그렇게 많은 관심을 받지는 못했다. (관심을 받지 못함에도 궁금한 것을 연구하는) 이러한 자세야말로 회사로 하여금 소프트웨어 개발에 있어 더욱 애자일 하고 비용 효율적인 방법을 찾도록 변화시키는 출발점이라 할 수 있다.

다른 사람에게 의미를 설명하기 가장 어려운 것들 중의 하나는, 이미 모든 사람들이 그것을 알고 있다고 생각하는 어떤 것들이다. 우리는 이미 어떻게 듣고, 어떻게 읽고, 어떻게 생각하며, 우리 삶의 일화를 어떻게 이야기 해야 할지 이미 알고 있다. 성인으로서 우리는 이런 일들을 매일 하고 있다. 그러나 일반적인 사람들이 소유하고 있는 어떤 레벨들의 기술들은, 어떤 특별한 경우에는 적용되기 힘들 때도 있다. 정신치료 상담사들은 노련한 듣기 전문가여야 하고, 변호사들은 읽기 전문가여야 할 것이다; 과학자들은 그들의 실수에 대한 고찰을 더 세심하게 정련할 필요가 있고 저널리스트들은 그들의 거실에서 오가는 잡담을 능가하는 이야기를 쓸 수 있어야 한다.

자 이제, 탐색적 테스팅(Exploratory Testing; ET)에 대해 알아보자: 이것은 **학습과 테스트 디자인 그리고 테스트 실행을 동시에 수행하는 것이다.** 이것은 매우 간단한 컨셉이다. 그러나 한 문장으로 어떤 개념을 표현할 수 있다는 것은, 더 이상 표현할 가치가 없는 아주 간단한 것처럼 보인다는 것도 사실이다. 탐색적 테스팅이 가지고 있는 상황에 따라 현저하게 변화하는 구조(highly situational structure)는 미숙한 관찰자들에게 탐색적 테스팅이 전혀 체계적이지 않은 것처럼 보이게 할 수도 있다. 이런 이유로 인해 몇몇 경우를 제외하고는, 소프트웨어 테스팅에 관한 글에서 탐색적 테스팅이 언급되지 않거나, 혹은 언급된다고 해도 단지 별 쓸모없는 행위로 잘못 이해되고 있는 것이다.

탐색적 테스팅은 또한 애드혹(ad hoc) 테스팅으로 알려져 있다. 불행하게도, 애드혹은 종종 내실이 없고 그리 신경을 쓰지 않아도 되는 작업과 동일한 의미로 사용되어왔다. 1990년대 초반, 일련의 테스트 방법론 연구진(최근에는 그들 스스로 **정황 주도적 학파(Context-Driven School)**라고 부른다)들은 애드혹을 대신해 "탐색적(exploratory)"이라는 단어를 사용하기 시작했다. 이 새로운 단어는 **Cem Kaner**의 책 **Testing Computer Software**에서 처음 사용되었으며, 스크립트화 되지 않은 테스팅을 포함하는 지배적인 사고 프로세스를 강조하고, 아울러 이러한 개념을 전파 가능하도록 하기 위해 고안된 개념이었다. 실제로, 탐색적 테스팅은 다른 지적인 행위와 마찬가지로 잘 훈련되고 체계화될 수 있다. 마이크로소프트는 윈도 호환성

[1] 역자 주: "Exploratory"라는 단어의 해석으로 주로 '탐험적', 혹은 '탐색적' 이라는 용어가 사용되고 있다. 본고에서는 최근 STEN을 비롯한 웹상에서 주로 사용되고 있는 '탐색적'이라는 단어를 사용하기로 한다.

(http://www.satisfice.com/tools/procedure.pdf)을 위한 써드파티 애플리케이션의 인증을 위해 정형화된 탐색적 테스팅 방식을 사용하고 있으며, 세션 기반 테스트 관리 (http://www.satisfice.com/sbtm)는 탐색적 테스팅을 좀 더 큰 규모에서 감사 가능하고 측정 가능한 것으로 만들기 위해 특별히 설계된 것이다.

우리가 탐색적 테스팅 이라는 단어를 사용할 때, 어떤 면에서도 변형되거나 왜곡된 의미로 사용하는 것이 아니다. 우리는 이 단어를 정규적인 사전적 의미 그대로 사용하는 것이며, 단지 테스팅이라는 분야에 응용해 사용하는 것 뿐이다. "탐색적"이라는 단어를 사용하는 경우는 지질학에서 이 용어를 사용하는 경우와 동일한데, 특히 18세기와 19세기에 걸쳐 왕립 지질학 학회의 탐험가들 사이에서 방법론적으로 유사한 문제가 제기되어 논쟁거리가 되었다:

"여행이 탐색으로 인정받기 위해서는 그 여정을 신뢰할 수 있어야 하며, 고난과 위험을 포함하고, 또한 발견의 경이로움도 포함하고 있어야 한다. 그러므로, 크리켓과 같이, 충분한 경험을 가지지 못한 사람들에게는 설명하기 어려운 무언가가 있다. 그러나 하나의 요소만은 확고하다; 실제로 탐색의 시대는 앞선 발견의 시대와는 확연히 구별되고, 아울러 '탐색(Exploration)'이라는 단어를 필연적으로 사용해야 한다. 이것은, 아주 간단하게, 과학에 대한 경외라고도 할 수 있다."

- John Keay, 탐색에 관한 영구불변의 책(The Permanent Book of Exploration)

별빛이 새벽녘 햇빛에 아스러지지만, 적외선의 영역에서는 아직도 밝게 타고 있는 모습을 볼 수 있듯이, 탐색적 테스팅에 대한 간단한 아이디어는 ***기술(Skill)***이라는 스펙트럼 하에서 조망될 때 더 흥미롭고 정교해 질 수 있다. 체스를 가정해보자. 체스를 플레이하는 프로시져는 사실 기술보다는 훨씬 흥미로울 게 없다. 아무도 에마누엘 라스커(Emanuel Lasker)가 1894년에 슈타인츠(Steintz)를 물리쳐 챔피언의 자리에 오를 때, 얼마나 환상적인 순서를 따라 체스를 두었는지에 대해서는 이야기하지 않는다. 체스를 두는 프로시져는 거의 변하지 않는 것이고, 변화의 선택에 불과하다. 아울러 이것은 다음에 무엇을 움직일 것인가 하는 플레이어의 기술의 하나일 뿐이다. 탐색적 테스팅을 더 흥미롭게 만드는 것은, 나의 관점에서는 무척이나 중요한 것인데, 테스터들이 정확하고 효율적으로 ***듣고, 읽고, 생각하고 보고하는 능력***을 소유하고 있다면, 규정되어 있는 어떠한 절차도 사용하지 않더라도, 테스팅에 대한 탐색적인 접근법은 미리 규정된 다양한 어떤 방법들보다 몇 배나 더 효율적(중요한 정보를 드러낸다는 측면에서) 일 수 있다는 것이다. 심지어 특별한 기술을 보유하지 못한 테스터들일지라도, 적절하게 통제 받고 관리된다면, 스크립트를 사용했을 때는 예상하지 못했던 유용한 결과를 도출해 낼 수 있다. 다시, 역사에서 이와 비슷한 개념을 찾아본다면, 경이로울 정도의 성공을 거두었던 루이스(Lewis)와 클라크(Clark)의 탐색[2]이야말로 탐색에 있어서 기술적 모델의 훌륭한 예제가 될 것이다:

[2] 역자 주: 메리웨더 루이스(Meriwether Lewis)와 윌리엄 클라크(William Clark)의 서북 공정을 일컫는 것으로 미국 최초로 육로를 통해 태평양 연안까지 진출했다가 돌아온 탐색이다. 토머스 제퍼슨 대통령에 의해 준비된 이 탐색으로 인해 북미의 동부 지역에만 제한되어 있던 미국이 서부로 본격적인 관심을 돌리게 되는 계기가 된다.

"루이스는 외교적이고 상업적인 사상가였으며, 클라크는 협상가였다. 루이스는 필라델피아에서 식물학, 동물학, 천체 항해를 수학한 과학 전공자였으며, 클라크는 기술자 겸 지리학자면서 아울러 첨단 공예의 뛰어난 장인이기도 했다... 두 사람 모두 탁월한 지성을 겸비하고 있었다. 북아메리카의 탐험 역사를 통틀어봐도 그들만큼 지성적인 사람을 찾아보기 힘들었다."

- Bernard De Voto, 루이스와 클라크에 대한 저널(The Journals of Lewis and Clark)

자, 이제 잠시, 이들 선구자들로부터 주제를 다시 되돌려보자. 물론, 테스트는 여러 가지 다양한 이유에서 반복적인 스크립트화된 양식을 줄여야 할 필요가 있다. 당신은 특정한 작업, 즉 매번 같은 방식으로 수행되는 벤치마크와 같은 종류의 테스트를 책임지고 있을 수 있다. 탐색적 테스팅은 스크립팅 이라는 개념과 상반되는 개념이 아니다. 어떤 상황에서는, 더 스크립트화된 방법을 통해서 당신의 테스트 미션을 달성할 수도 있을 것이다; 또 다른 상황에서는, 당신의 미션은 테스트를 수행하면서 당신이 직접 테스트를 생성하고 개선하는 바로 그 능력으로부터 더 큰 도움을 받을 수도 있을 것이다. 따라서, 나는 대부분의 경우 스크립트화된 방식과 탐색적 방식을 같이 사용하는 것이 유리하다는 것을 발견했다.

탐색적 테스팅 정의

나는 수 차례 탐색적 테스팅을 정의하려고 노력해왔다. 도출된 여러 가지 정의 중, 나의 동료들이 가장 선호하는 것은 아래와 같다:

탐색적 테스팅은 학습과, 테스트 디자인, 그리고 테스트 수행을 동시에 하는 것이다.
(Exploratory testing is simultaneous learning, test design, and test execution.)

즉, 탐색적 테스팅은 테스터가 테스트가 수행되는 동안 능동적으로 테스트 디자인을 컨트롤하고, 테스트로부터 도출된 정보를 사용해 더 새롭고 발전된 테스트를 디자인하는 것이다.
탐색적 테스팅의 이러한 관점에서 본다면, 사람이 수행하는 모든 테스트는 사실상 ***어느 정도까지는*** 탐색적이라고 말할 수 있다. 이러한 관점은 탐색적 테스팅을 훌륭한 테스터들이 수행하는 많은 행위들을 담고있는 사고 프로세스(Thought process)로 다루고 있으며, 미리 작성되는 **완전히 스크립트화된 테스팅**과, 테스트를 수행할 때마다 테스트 아이디어가 발생하는 **완전한 탐색적 테스팅**의 중간 단계라고 볼 수 있다. 이러한 연속성 때문에, "탐색적 테스팅을 수행하는가?"라는 질문은 "어떤 방법으로, 그리고 어느 정도로 탐색적으로 테스팅을 수행하는가?"라고 바꾸는 것이 더 적합해 보인다.

이 글에서 내가 탐색적 테스팅을 언급하고 별도로 정의하지 않을 때는, 완전한 스크립트 테스팅보다는 오히려 완전한 탐색적 테스팅에 가까운 것을 칭하는 것이다. 즉, 테스팅 이라는 단어 자체가 이미 실질적으로 어느 정도 그 정의를 만족시키고 있는 것이다. 스크립트가 없거나 별도의 지시가 없는 상태에서 완전한, 혹은 완전한 탐색적 테스팅에 거의 가까운 테스팅을 언급할 때는, 이

것을 ***프리스타일*** 탐색적 테스팅이라고 부를 것이다.

탐색적 테스팅은 상황과 깊이 관련된 행위이다
(Exploratory Testing is a Profoundly Situational Practice)

직소 퍼즐을 풀어본 적이 있는가? 만약 그렇다면, 당신은 이미 탐색적 테스팅을 경험해 본 것이다. 그 과정에 무슨 일이 벌어지는지 생각해 보자. 당신은 퍼즐의 한 조각을 들고 이것이 어디에 맞는지, 서로 연결되지 않은 조각들이 뭉쳐있는 곳을 훑어볼 것이다. 새로운 조각을 들고 한 번 쳐다보는 것이 하나의 테스트 케이스이다("이 조각이 저 조각과 연결될까? 아닌가? 이걸 뒤집어보면 어떨까? 글쎄, 거의 맞기는 한데 그림이 맞지 않네..."). 당신은 네 귀퉁이의 조각을 먼저 맞춘다던가, 아니면 중앙 부위의 그림을 먼저 맞추거나, 혹은 박스에 그려져 있는 그림의 어떤 속성을 가지고 좀 더 정밀하게 직소 테스팅 프로세스를 진행할 수도 있을 것이다. 당신이 퍼즐을 조립하기 전에, 혹은 퍼즐을 완성한 후의 그림 종류가 어떤 것인지를 전혀 알지도 못한 채 당신의 모든 직소 "테스트 케이스"를 문서화하고 디자인 한다는 것을 상상할 수 있는가?

만약 내가 직소 퍼즐을 풀어야 한다면, 나는 내가 퍼즐에 대해 경험한 것들과 그림의 형태대로 작업 방식을 바꿀 것이다. 만약 내가 어떤 한 색깔로 이루어진 부분을 맞추려고 한다면, 그 색깔과 유사한 조각들을 한 군데에 모을 것이다. 만약 내가 특별히 구별되는 모양을 맞추려고 한다면, 그 모양들을 한 군데에 모을 것이다. 만약 내가 잠시 동안 하나의 테스트에만 매진해야 한다면, 나는 다른 종류의 테스트는 잊어버리려 할 것이다. 만약 내가 어느 정도 충분한 크기의 조각들을 맞췄다면, 나는 어디가 다른 부분과 연결되는지 알기 위해 퍼즐의 프레임 부분을 살펴볼 것이다. 때로는 나 스스로 이 방식이 무척이나 체계화되지 않았다고 여길 수도 있다. 그럴 땐 다시 지나왔던 단계들을 되돌아보고, 상황을 분석해 좀 더 체계적인 계획을 채택한다. 어떻게 프로세스가 흘러가고, 각각의 순간들이 어떻게 연속성을 가지고 있으며, 수행자의 통제 하에 놓여져 있는지에 주의를 기울이라. 이런 과정들은 직소 퍼즐을 풀어가는 것과 무척이나 흡사하지 않은가? 만약 그렇다면, 이러한 사고 과정을 세심하게 미리 문서화 한다는 것이 무척이나 불합리하다는 것에도 동의할 수 있을 것이다. 아래에 이어지는 명백한 사항들 중의 하나를 줄여나가는 것만이 우리의 작업 속도를 늦춰줄 것이다.

이것은 퍼즐에 대한 일반적인 지침이다: 퍼즐은 무엇을 해야할 지 헷갈려 하는 사람들을 변화시킨다(the puzzle changes the puzzling). 퍼즐의 명세(Specifics)는 퍼즐을 풀어나가는 도중에 발생하며, 우리가 퍼즐을 풀어나가는 전략에 영향을 끼친다. 이러한 사실은 어떤 탐색적인 조사에서도 핵심사항이며, 테스팅, 개발, 혹은 심지어 과학적인 연구나 발견 작업에서도 동일하다.

어떤 종류의 명세가 탐색적 테스팅에 영향을 끼치는가? 여기 그 중 몇 가지 예를 들어보자:

- 테스트 프로젝트의 미션
- 이번 특정한 테스트 세션의 미션
- 테스터의 역할
- 테스터(스킬, 재능, 그리고 취향)

- 활용 가능한 툴과 장비
- 활용 가능한 시간
- 활용 가능한 테스트 데이터와 재료
- 활용 가능한 다른 사람들로부터의 도움들
- 책임 요구사항(Accountability Requirements)
- 테스터의 고객이 유의하는 것들(What the tester's clients care about)
- 현재의 테스팅 전략
- 동일한 프로젝트의 다른 테스트 업무 현황
- 제품, 아울러 제품의
  - 유저 인터페이스
  - 행동(Behavior)
  - 실행 현황(its present state of execution)
  - 결함들
  - 테스트 가능성(혹은 용이성; Testability)
  - 목적
- 테스터가 제품에 대해 알고 있는 것들
  - 앞선 테스트에서 발생한 일들(What just happened in the previous test)
  - 제품에서 알려진 문제점들
  - 과거의 문제점들
  - 장점과 약점
  - 리스크 영역과 인지된 리스크의 정도
  - 최근의 변경 사항들
  - 직접 관찰한 것들
  - 이것에 관한 소문들
  - 사용자와 사용자 행위의 본질들(The nature of its users and user behavior)
  - 어떻게 동작하는가(How it's supposed to work)
  - 어떻게 통합되는가(How it's put together)
  - 다른 제품과 어떤 것이 다르고 유사한가(How it's similar to or different from other products)
- 테스터가 제품에 대해서 무엇을 알고 싶어 하는가

어떤 테스트를 설계했는지를 묻는 대신에, 탐색적인 테스터는 ***'내가 지금 수행할 수 있는 최상의 테스트는 무엇인가?'***를 묻는다. 각각의 고려사항들은 어떤 테스트가 필요한 지에 대해 영향을 끼칠 수 있다. 이러한 요인들은 테스트 프로젝트가 진행되는 동안 지속적으로 변화하며, 심지어는 테스트 세션의 중간중간에도 끊임없이 변경된다. 탐색적 테스팅의 위력은 테스트 프로세스를 거치면서 최적화되지만, 반면 스크립트는, 그들이 변화하지 않기 때문에, 시간이 지날수록 그 위력이 약해지는 경향이 있다. 스크립트 테스트의 위력이 점점 약해지는 데에는 여러 가지 이유가 있는데, 그 중 가장 주요한 원인은 스크립트화된 테스트를 한 번 수행하고 나서 문제를 발견하지

못한 상태에서, 두 번째로 스크립트를 실행할 경우, 문제를 찾을 수 있는 기회가 대부분의 경우 새로운 테스트를 수행하는 것보다도 실제로 무척이나 낮다는 것이다.

탐색적 테스팅 수행하기(Practicing Exploratory Testing)

탐색적 테스팅의 외적인 구성 요소는 설명하기에 매우 용이하다. 일정 ***기간*** 동안, ***테스터***가 테스팅이라는 ***목적***을 만족시키기 위해 ***제품***과 상호작용을 하고, 그 결과를 ***보고***한다. 여기에 탐색적 테스팅의 기본적인 외적 구성 요소가 모두 제시되고 있다: 시간, 테스터, 제품, 목적, 그리고 보고가 그것이다. 우리 스스로를 목적과 동화시키고, 제품에 대한 질문을 도출하고 그것에 대한 답변이 가능하다면, 우리 스스로가 우리의 목적에 만족할 수 있을 것이다. 아울러 이러한 질문에 답변하기 위해 테스트를 디자인하고, 테스트를 실행함으로써 답변을 얻는 반복적인 사이클을 통해 최종적인 목적이 충족되는 것이다. 종종 우리가 수행하는 테스트가 그 질문에 대한 만족할만한 답변을 주지 못하지만, 그래서 우리는 테스트를 조정하고 계속 노력해야 하는 것이다(즉, 탐색해야 한다는 것이다). 우리는 언제라도 우리의 현 상황과 결과들을 보고할 준비가 되어있어야 한다.

탐색적인 테스트 세션은 종종 그 목적과 아마도 사용될 몇 가지 전략이 언급되어 있는 차터(Charter)[3]로부터 시작된다. 차터는 테스터 스스로가 선택할 수도 있고, 그렇지 않을 경우 테스트 리더나 매니저가 할당한다. 때때로 차터는 (텍스트 형식으로) 기술된다. 몇 몇 조직에서는, 테스트 케이스와 프로시져가 매우 모호하게 기술되어 있어, 탐색적 테스팅에서 차터가 하는 역할 정도만 수행하는 경우도 있다.

아래에 의사 결정 분석 제품인 DecideRight[4]를 위한 몇 가지 테스팅 차터의 예제가 있다:

- DecideRight의 제품 요소를 탐색하고 분석하라. 테스트 커버리지 아웃라인을 만들어라.
- DecideRight 매뉴얼에 있는 모든 요구사항(Claim)을 구별하고 테스트하라.(프린트된 매뉴얼에 체크마크/X/? 등의 기호로 표시하거나 각각 테스트된 요구사항들을 당신의 노트에 기록하라)
- DecideRight의 워크 플로우를 정의하고 각각을 실행해보라. 플로우는 사실적인 사용 시나리오를 반영하고 있어야 하며, 포괄적으로 제품의 우선되는 기능들을 모두 포함하고 있어야 한다.
- 우리는 DecideRight의 의사 결정 복잡도를 점점 증가시킴으로써, DecideRight의 퍼포먼스(Performance)와 신뢰성(Reliability)이라는 속성을 이해할 필요가 있다. 일반적인 시나리오로부터 시작해서, 애플리케이션이 행(Hang) 되거나 충돌이 발생하거나, 아니면 자연스럽게 사용자로 하여금 무리하게 애플리케이션을 사용하는 것을 멈추도록 하는 옵션과 요소

[3] 역자 주: 아래에 "Sometimes charters are written down" 이라는 문장이 이어지는 것으로 보아 저자가 말하는 "Charter"는 개략적인 문서 형태의 예비 문서, 혹은 작업 정의서 정도로 보는 것이 타당할 듯 하다. "~written down"이라는 말은 좀 더 정형화된 문서로 기록된다는 것을 뜻하는 듯 하다.

[4] 역자 주: "DecideRight"라는 일종의 소프트웨어 제품을 뜻하는 듯하다.
제조사 홈페이지(http://www.skyhunter.com/dr.html) 참조.

들을 추가시켜 보라.

- 데이터 입력이 허용하는 모든 필드를 테스트 해보라(당신은 이미 그 방법을 알고 있을 것이다: 기능(Function), 스트레스(Stress), 그리고 제한(Limits)).
- DecideRight 시나리오의 파일 포맷을 분석하고, DecideRight의 구성 요소들이 프로그램 상에서 조작될 경우 애플리케이션이 취해야 하는 행동들을 결정하라. 에러 핸들링을 테스트하고 잘못된 시나리오 파일을 사용했을 때의 퍼포먼스를 체크하라.
- UI가 윈도 인터페이스 표준을 지키는지 체크하라.
- 시나리오 파일에 오류를 발생시키는 방법이 있는가? 오류가 발생했는지 어떻게 알 수 있는가? 자동 파일 체커(Automatic file checker)의 작성이 가능한지 조사해보라. 개발자가 이미 그런 검사들을 완료했는지 조사해보라.
- 외부 애플리케이션, 특히 마이크로소프트 워드와의 통합을 테스트 해보라.
- 실험을 통해 의사 결정 분석 알고리듬을 결정하고 이를 액셀에서 다시 수행해보라. 그 다음, DecideRight에서의 복잡한 의사 결정 시나리오를 테스트 하기 위해 스프레드 시트를 사용해 보라.
- AppVerifier[5]를 구동한 환경 하에서 DecideRight를 실행하고 에러를 보고하라.

만약 당신이 이 차터에서 애매모호 하거나 잘 모르는 것이 있다고 판단해도, 그리 놀랄만한 일이 아니다. 이것들은 조직 내에서 사용되는 기대값, 용어, 기법 그리고 툴 등으로 잘 훈련된 테스터들에게, 간결하고 명확하게 테스트 세션의 목적을 알려주기 위해 만들어진 것이다. 기억하라, 문서로 작성된 모든 행위를 하려고 시도하는 것보다, 탐색적 테스팅을 사용함으로써 우리는 우리의 스킬을 최대한 활용할 수 있는 것이다.

프리스타일 탐색적 테스팅에 있어서, 탐색적 테스팅의 결과로 도출되는 단 하나의 공식적인 결과는 버그 리포트 뿐이다. 세션 기반의 테스트 관리에서, 각각의 탐색적 테스팅 세션은 테스트 리더에 의해 리뷰된 문서 기록을 그 결과로 남긴다. 또한 업데이트된 테스트 재료, 혹은 새로운 테스트 데이터들을 그 결과로 남기기도 한다. 가장 공식적으로 기록된 테스트 프로시져들은, 아마도 어떤 종류의 탐색적 테스팅 프로세스를 통해 만들어졌을 것이다. 탐색적 테스팅에 있어 외부 트래핑, 입력값 및 출력값도 검토해 볼 가치가 있지만, 가장 중요한 것은 탐색적 테스트의 내부적 구조, 즉 테스터의 머리 속에서 일어나는 것들이다. 그것이 탐색적 테스팅이 성공하거나 실패하는 이유이다; 탁월한 탐험가들은 아마추어와는 구별되기 마련이다. 이것은 복잡한 문제지만, 탁월한 탐색적 테스팅을 위한 몇 가지 기본적인 사항들이 있다:

- ***테스트 디자인(Test Design):*** 탐색적인 테스터들은 첫 째로 그리고 무엇보다도 테스트 디자이너들이다. 누구나 우연히 테스트를 디자인 할 수는 있지만, 훌륭한 탐색적 테스터는 제품을 체계적으로 탐색할 수 있는 테스트를 정교하게 만들 수 있다. 이는

[5] 역자 주: AppVerifier는 윈도 API를 후킹해 런타임 시에 발생하는 문제점을 예측해 주는 프로그램이다. http://www.serious-code.net/moin.cgi/AppVerifier

제품을 분석하고, 위험을 평가하고, 툴을 사용하고, 비평적으로 사고하는 것과 같은 기술들을 요구한다.

- ***신중한 관찰(Careful Observation)***: 훌륭한 탐색적 테스터는 초심자들보다 월등히 뛰어난 신중한 관찰자이며, 이러한 이유로, 경험있는 스크립트 테스터보다도 월등하다. 스크립트 테스터는 단지 스크립트가 그에게 관찰하라고 지시한 것만을 관찰한다. 탐색적 테스터는 일반적이지 않거나 이해할 수 없는 모든 것들을 관찰해야 한다. 탐색적 테스터는 압박을 받는 상황에서도 추론으로부터 관찰을 구별하는 데 신중해야 하며, 중요한 테스트나 제품의 행동에 대해 선입견을 가지게 하는 어떤 예상된 가정도 하지 않아야 한다.
- ***비판적인 사고(Critical Thinking)***: 훌륭한 탐색적 테스터는 그들의 이론을 리뷰하고 설명할 수 있어야 하며, 그들 스스로 자신의 사고에서 오류를 발견해야 한다. 이것은 특히 탐색적 테스트 도중 현재의 상황을 보고하거나, 결함을 조사할 때 무척이나 중요하다.
- ***다양한 생각들(Diverse ideas)***: 훌륭한 탐색적 테스터들은 초심자들보다 더 나은 생각을 많이 만들어낸다. 그들은 이를 위해 발견적 학습법(Heuristics)을 사용하는 경우도 있다. 발견적 학습법이란 가이드라인, 일반적인 체크 리스트, 암기법, 혹은 경험에 바탕을 둔 방법(Rules of thumb)와 같은 정신적 수단을 의미한다. Satisfice Heuristic Test Strategy Model(http://www.satisfice.com/tools/satisfice-tsm-4p.pdf)은 다양한 생각들을 빠르게 산출할 수 있는 발견적 학습법의 예제라고 할 수 있다. 제임스 휘태커(James Whittaker)와 알란 요한센(Alan Jorgensen)의 "17 attacks"는 또 다른 예라고 할 수 있다. 테스터의 다양한 기질과 팀의 백그라운드 역시, 탁월한 탐색적 테스터가 그룹 브레인스토밍 프로세스를 통해 더 나은 테스트 아이디어를 도출하는데 유용하게 활용될 수 있다.
- ***풍부한 자원들(Rich Resources)***: 훌륭한 탐색적 테스터는 많은 양의 툴 목록, 정보 제공처, 테스트 데이터, 그리고 믿을 만한 동료들을 만들어낸다. 테스팅을 하는 동안, 그들은 이들 자원들을 테스팅에 직접 적용할 수 있는 기회를 유심히 찾아본다.

탐색적 테스팅 관리(Managing Exploratory Testing)

많은 조직에서 테스트 매니저와 테스트 리더를 구별하는 것은 무척이나 중요한 일이다. 테스트 매니저는 일반적으로 직원들의 임용과 해고 권한을 가지고 있으며, 다른 행정적인 책임도 가지고 있는 반면, 테스트 리더는 단지 테스트 전략과 전술에만 역량을 집중한다. 탐색적 테스트 관리를 논함에 있어, 나는 테스트 매니저가 비록 그 역할을 수행할 수 있음에도 불구하고, 테스트 리더라는 단어를 사용할 것이다.

프리스타일 탐색적 테스팅은 크게 두 가지 방식으로 관리될 수 있다: **'위임(Delegation)'**과 **'참여(Participation)'**가 그것이다. 위임 방식에서 테스트 리더는 차터를 작성한다. 그러면 테스터들은 그들의 일, 즉 차터를 작성하기 위해 테스트를 디자인하고 수행하며, 그 결과를 보고한다. 실제로는 한 테스터가 하나의 컴포넌트 셋을 계속해서 할당 받는 것이 관례이며, 이로 인해 테스터가 지속적인 학습 효과를 가지고, 이로 인해 전체 프로젝트가 훨씬 효율적으로

진행될 수 있다. 테스트 보고는 문서화되거나 혹은 구두로 전달될 수 있다. ***Cem Kaner***는 테스트 프로그레스를 논의하기 위해 일주일에 적어도 한 번 정기적인 회의를 할 것을 제안한다. 그는 표준적인 질문인 "당신이 최근에 찾아낸 버그 중에 가장 흥미로운 것은 무엇인가요? 보여주세요."라는 명제를 가지고 회의를 하는 것이 유용하다는 것을 발견했다. 세션 기반 방법에서, 테스트 보고는 기록되고, 테스터는 매일 한 번 이상 인터뷰된다.

위임에 의한 관리는 본질적으로 각각의 테스터들을 그들 자신의 시간과 업무에 대한 가치를 관리하는 한 명의 임원으로 본다. 임원의 한 사람으로서, 아울러 생산적이고 책임감 있는 테스터로 신뢰받지 못한 테스터는 많은 부분을 할당 받지 못하며, 짧은 탐색적 세션에 참가하는 것으로 그 역할을 제한받고, 다른 사람들에게 좀 더 신중해야 하는 사람으로 치부된다. 탐색적 테스터들의 리더가 된다는 것은, 반독립적인 창조적인 구성원들의 코치가 된다는 것을 의미한다.

참여에 의한 관리는 테스트 리더가 나머지 테스터들과 함께 일을 하면서 테스팅을 주도하는 것이다. 실제로, 이 방법은 다른 사람들에게 행정적인 권한과 회의 참여 책임을 위임하지 않는 한, 가장 최선의 방법이다. 참여는 리더들에게 테스트 전략을 실시간으로 수립할 수 있도록 하고, 지속적으로 그가 팀에 무엇을 바라는지 설명할 수 있도록 한다. 테스트 매니저는 팀의 성과에 대한 최종적인 책임을 지는 만큼, 참여에 의한 관리는 관리자가 이러한 책임을 질 수 있는 적절한 포지션을 제공한다. 탐색적 테스팅을 수행하는 동안의 혼란과, 테스팅이 불충분하게 수행될 가능성과 같은 다양한 문제들이, 테스터 리더가 테스팅에 밀접하게 참여함으로써 상당 부분 해소될 수 있다.

대부분의 테스트 리더들이 그들의 테스트를 조직화 하기 위해 일정한 종류의 테스트 커버리지 가이드를 사용한다. 이러한 가이드는 테스트 커버리지 아웃라인이나 메트릭스(Matrix), 위험 목록(A list of risks), 혹은 심지어 유행이 지난 To Do 리스트의 형태로 존재할 수 있다.

팀 단위의 탐색적 테스팅은 매우 강력한 기법이다. 실제로 이를 수행해 본 많은 테스트 리더들이, 사람들이 함께 일함으로써 생기는 사회적 에너지(Social energy)와, 동시에 같은 장비를 사용해 버그를 도출하는 것들이, 동일한 사람들이 독립적으로 업무를 진행할 때보다 더 나은 아이디어들을 도출해 낸다는 것을 경험했다. 탐색적 테스팅 팀을 조직하는 방법 중의 하나로, 테스터를 둘씩 짝 지어서 테스트를 할 때 하나의 컴퓨터를 공유하도록 하는 방법이 있다. 내가 수행했던 또 다른 방법은 한 테스터가 키보드를 가지고 "드라이브(Drive)"하는 동안 다른 사람들은 그것을 보면서 코멘트 하는 것이다. 드라이빙 테스터가 문제를 발견하거나 논의할 필요가 있는 의문점을 제기하면, 관람자 중의 한 사람이 관람을 중단하고 다른 테스트 플랫폼을 사용해 그 문제를 재현하고 관찰하려고 시도한다. 이러한 방법은 드라이빙 테스터가 최소한의 간섭을 받으면서 테스팅의 주된 작업을 지속할 수 있도록 한다. 이 방법은 블록버스팅 툴을 사용해 일반적일 루트를 벗어난 테스트를 시도하거나, 테스터들을 제품에 사용된 기술에 대해 훈련할 때, 혹은 테스트 디자인 기법에 대해서 훈련할 때 특히 유용하게 사용된다.

탐색적 테스팅이 적합한 곳(Where ET Fits)

일반적으로 탐색적 테스팅은 다음 테스트가 어떤 것이 수행될지 명확하지 않을 때, 혹은 당신이 늘 수행하던 뻔한 테스트를 뛰어넘는 어떤 다른 테스트를 원할 때 수행된다. 특히, 프리스타일 탐색적 테스팅은 다음과 같은 상황에 적합하다:

- 새로운 제품이나 기능에 빠른 피드백을 제공해야 할 필요가 있을 때
- 제품에 대해 신속하게 학습해야 할 때
- 스크립트를 사용해 이미 테스트를 완료하고 좀 더 심도 있는 테스팅이 필요할 때
- 짧은 시간 안에 가장 중요한 단일 버그를 찾고 싶을 때
- 간단한 독립적인 조사를 수행함으로써 다른 테스터의 작업을 체크하고 싶을 때
- 특정한 결함을 조사하고 격리시키길 원할 때
- 특정한 위험이 발생한 영역에 대해 스크립트 테스트가 필요한지 평가하기 위해 특정한 위험의 상태를 조사하길 원할 때

프리스타일 탐색적 테스팅은 제쳐두고, 탐색적 테스팅은 사전에 완벽하게 기술되지 않은 모든 테스팅이 수행되는 그 어떤 곳에도 적합하다. 이는 모든 상황을 포함하는 것이며, 여기에 더해 아래의 추가적인 상황들도 최소한 하나 이상 추가될 수 있다:

- 스크립트 테스트를 즉시 수행할 때
- 막연해 보이는 테스트에 대한 설명이 필요할 때
- 제품 분석과 테스트 계획
- 현존하는 테스트의 개선
- 새로운 테스트 스크립트 작성
- 오래된 버그 리포트를 기반으로 하는 리그레션 테스팅
- 유저 매뉴얼을 읽는 것과 각각의 항목을 체크하는 것에 기반을 둔 테스팅

탐색적 테스팅은 매우 강력한데, 그 이유는 정보가 테스트 수행으로부터 테스트 디자인까지 거꾸로 순환되기 때문이다. 피드백 순환이 약해지거나, 혹은 그 순환이 길어지거나, 느려지거나, 혹은 비용이 많이 들게 된다면, 탐색적 테스팅은 그 위력을 잃어버리고 만다. 만약 그렇게 된다면, 우리는 조심스럽게 스크립트 테스트로 되돌아가야만 한다. 스크립트 테스트를 사용해야 하는 부분은 아마도 특별히 논쟁의 여지가 있거나, 혹은 높은 수준의 관리 혹은 사용자들의 동의가 필요하다고 간주되는 부분일 것이다. 이런 경우 탐색적 테스팅과 스크립트 테스팅 전략을 같이 사용함으로써, 두 기법으로부터 최상의 효과를 함께 얻을 수 있을 것이다.

실무에서의 탐색적 테스팅(ET in Action)

나는 네 시간 동안 인기 있는 사진 편집 프로그램을 테스팅 해야 하는 미션을 받은 적이 있다. 나의 미션은 이 제품이 마이크로소프트 윈도 호환성 인증 프로그램의 표준을 준수하는지 평가하는 것이었다. 그 테스트를 수행하는 프로시져는 매우 정형화된 탐색적 테스팅 프로세스로

꾸며졌다. 나의 목표는 호환성 요구사항을 위배하는 모든 사항을 찾아내고 그것들을 명확하게 문서화 하는 것이었다.

이렇게 명확하지 않은 차터를 염두에 두고, 테스트를 시작했다. 가장 간단한 탐색의 경험칙 중 하나를 반영시켜, 나는 애플리케이션의 메뉴를 하나 하나 작동해보기 시작했다. 그것을 수행하면서 나는 제품의 우선적인 기능에 대한 아웃라인을 만들어가기 시작했다. 이것이 나중에 내가 무엇을 테스트 하고, 무엇을 테스트 하지 않았는지에 대한 보고의 기본이 되는 것이다.

나는 Save As... 라는 기능이 사용자들로 하여금 다양한 이미지 품질 속성을 조절할 수 있게 하는 매우 정교한 여러가지 컨트롤을 제공한다는 것에 주목했다. 나는 이미지 품질에 대한 기술적인 속성을 알지 못했기 때문에, 스스로 이러한 기능에 대한 테스트를 할 준비가 되지 못했다고 생각했다. 대신, 나는 이슈를 기입하고 이미지 품질에 대한 문서를 공부해 이를 테스트 전략에 반영할 수 있도록 테스팅에 시간을 더 할애해 줄 수 있는지 여부를 고객에게 묻는 노트를 작성하기 시작했다. 나는 나의 노트를 작성하면서, 메뉴에 대한 작업을 지속했다.

탐색적 테스팅의 기본 전략은 일반적인 기본 계획을 가지고 있으면서도 당신 스스로 이들을 짧은 시간 안에 변형할 수 있도록 허용하는 것이다. **Cem Kaner**는 이것을 **"관광 버스(Tour Bus)" 원리**라고 부른다. 관광 버스에 탑승하고 있는 사람들은 필요할 때마다 차량에서 내려서 주위를 둘러본다. 핵심은 전체적으로 관광할 거리들을 놓치지 않으면서도, 버스에서 잠들지 않는 것이다. 메뉴에 대한 나의 관광을 중단하게 만든 첫 사건은 애플리케이션에 사용되는 메모리의 총량을 조절하는 다이얼로그 박스를 찾아냈을 때 발생했다. 이것은 나에게 즉각적으로 어떤 아이디어를 주었다(갑작스런 아이디어가 탐색적 테스팅 도중에 발생한 것이다). 윈도 호환성 프로그램의 요구사항 중의 하나가 안정성(Stability)이기 때문에, 제품이 최소한의 메모리를 사용하도록 해보는 것이 유용할 것이라고 생각했으며, 따라서 나는 메모리와 관련되어 있는 기능들을 집중적으로 수행해 보았다. 나는 시스템 메모리 슬라이드 바를 5%로 설정하고, 이미지 속성 결정 항목에서 이미지 사이즈를 100 평방인치(100inch X 100inch)로 설정해 보았다. 정말 커다란 캔버스다. 그 다음 이 캔버스를 자주색 점으로 가득 채우고 효과 메뉴로 가 몇 가지 특별한 그래픽 효과를 적용해 보았다.

오케이, 여기가 중요한 부분이다: "잔물결(Ripple)" 효과를 메뉴에서 선택하자, "밤(bam)"소리와 함께 제품이 순식간에 그 동작을 위한 메모리가 부족하다는 내용의 에러 메시지를 표시했다. 이것은 향후 이어질 테스트의 표준을 수립했다는 측면에서, 매우 흥미로운 행동이다. 이때부터 나는 앞으로의 행동에 대한 새로운 기대값을 가지게 된 것이다: 즉, '애플리케이션의 기능은 동작을 수행하기 위해 메모리가 부족할 경우, 스스로 수행을 방지할 수 있어야 한다'라는 표준을 가지게 된 것이다. 이것은 탐색적 테스팅에서 어떻게 하나의 테스트 결과가 다음 테스트에 영향을 주는가에 대한 완벽한 예제인데, 그 이유는 이후에도 나머지 메뉴들이 동일한 방식으로 동작하는지 확인하기 위해 다른 효과들을 지속적으로 조사했기 때문이다.
결과는? 내가 시험한 다른 어떤 것도 그런 식으로 동작하지 않았다. 대신, 5분 정도 어떤 일도

일어나지 않으며, 하드 디스크가 작동하는 것 외에는 다른 어떤 것도 관찰할 수 없었다. 마침내 **"Error -32: Sorry this Error is Fatal"**이라는 창이 뜨고 애플리케이션에서 크래쉬(Crash)가 발생했다.

이것은 매우 멋진 결과지만, 나는 대부분의 사용 가능한 메모리를 사용하기 위해 지속적으로 메모리 사용을 설정해 보지 않는 이상, 테스트가 완벽하게 끝난 것이라고 생각하지 않았다(탐색적인 테스터는 추후에 그들의 고객이 던질 수 있는 질문을 예상하려고 하므로). 놀랍게도, Error -32 메시지가 발생하는 대신, 전체 OS가 죽어버렸다. 윈도 2000이 그것을 지원하지 않았던 것이다. 이것은 단순한 크래쉬보다 훨씬 더 심각한 문제였다.

이 시점에서, 나는 나에게 주어진 4 시간의 프로세스 중 거의 30분을 소비하고, 애플리케이션이 호환성 인증을 받을 수 없도록 만드는 중요한 문제를 이미 찾아냈다. 이것은 좋은 뉴스다. 좋지 않은 뉴스는 시스템이 죽어버렸을 때 나의 테스트 노트를 잃어버렸다는 것이다. 리부팅 이후, 나는 내가 스트레스 테스팅 으로부터 많은 것을 배웠고, 메뉴 투어로 되돌아 가야 한다고 생각했다.

나는 이 테스트 스토리가 원론적이고 목적에 충실한 테스팅의 예제라고 생각한다. 나는 내가 커버한 테스팅 범위와, 내가 발견한 것들을 보고할 수 있었다. 또한 테스트를 부여 받은 목적과 연관시킬 수 있었다. 이러한 테스트 기법 역시 무척이나 반복적이다. 탐험적 테스팅은 최소한 대부분이 스크립트화 된 테스트만큼이나 반복적인데, 그 이유는 언제나 내가 무엇을 테스트 해야 하는지, 그리고 어떻게 테스트 해야 하는지에 대한 일관된 사고를 따르기 때문이다. 이러한 아이디어가 테스트 수행 도중 생겨났다는 사실은, 문서로부터 사고가 연유하지 않았다는 점에서 추상적이라고 할 수 있다. 나는 당신이 이것을 체계적이지 못한 테스팅(Unsystematic testing)과는 완전히 다른 것으로 보아주었으면 하는 바램이다. 만약 내가 나 스스로 수행한 테스트에 대해 자세하게 설명하지 못하고, 내가 무엇을 테스트 했고 나의 테스트 전략이 어떤 것이었으며, 테스팅과 미션을 연관 짓지 못했다면, 이것은 단지 체계적이지 못한 "애드혹(ad hoc)" 테스팅 이었을 것이다.

탐색적 테스팅의 생산성(The Productivity of ET)

탐색적 테스팅의 생산성을 나타내 주는 합리적인 수치는 없다; 탐색적 테스팅과 스크립트 테스팅을 비교해 그 테스팅의 생산성을 비교해 본 연구 사례가 아직 없기 때문이다. 우리가 가진 것이라곤 경험담들 뿐이다. 여기 나의 경험담 몇 가지가 있다.

나는 자동적으로 그들의 작업 로그를 남기는 시스템을 테스트 하는 탐색적 테스터들에게 테스팅 수업을 가르친 적이 있다. 그들은 거의 백여 개의 테스트를 디자인하고 수행해왔다. 동일한 시간 안에 그들에게 반복 가능한 테스트 프로시져를 작성하게 했을 때, 그들은 단지 겨우 한 개 정도를 작성할 수 있었다. 어떤 면에서는 탐색적인 테스트보다 반복적인 테스트 프로시져가 더 낫다고 주장할 수 있다. 이것은 그럴 수도 있고, 그렇지 않을 수도 있는 문제다. 개인적으로, 나는

작성된 테스트 스크립트를 활용하는 것이 탐색적 테스트보다 강력하지 않다고 생각한다.

나는 스크립트된 방법을 통해서 이미 테스트 된 애플리케이션을 테스팅 하는 팀을 이끈 적이 있고, 거기서 드라마틱한 문제를 찾아냈다. 어떤 사전 준비나 테스트 준비물도 없이, 나의 팀은 단지 10분만에 네트워크 장비를 충돌 나게 만든 적이 있었고 불행하게도 이로 인해 내부 하드 드라이브 포맷을 해야만 했다. 사실, 이 문제를 격리하고 조사하는 데 10분 이상의 시간이 소요되었지만, 이것은 처음에 문제가 발견되었다고 가정한다면, 스크립트된 테스트 프로시져에서도 10분 이상의 시간이 소요되었을 것이다.

나는 어느 대규모의 회사에서 스크립트 테스터들에게 둘러 쌓여 이제 갓 업무를 시작하는 탐색적 테스트 팀을 도와준 적이 있다. 그들은 테스트 산출물과 관련된 유지 보수 작업에서 자유로운 탐색적 테스터들이 스크립트 테스터들보다 더 많은 것을 주도하려 하고, 위험과 관련된 더 많은 소문을 추적하려 한다는 것을 발견했다. 그들이 더 많은 버그를 발견하고, 더 중요한 버그를 발견했다. 3년이 지난 다음, 그 팀은 아직도 제자리를 지키고 있다.

한 사람이 어느 컨퍼런스에서 나에게 다가와 대규모의 텔레콤 회사에서 탐색적 테스트 팀을 운영하고 있다고 말했다. 그는 그의 팀이 프로젝트에서 프로젝트로 옮겨 다니는데, 그가 이런 작업을 지속할 수 있는 이유는, 바로 그들의 메트릭이 주어진 시간 안에 스크립트된 테스트를 수행하는 사람들보다 4배나 많은 문제를 찾아낸다는 것을 보여주기 때문이라고 말했다.

이런 일화들이 어떤 사실을 증명해 주는 것은 아니다. 나는 단지 당신의 흥미를 돋우기 위해 이런 일화들을 이야기 한 것이다. 사실, 생산성이라는 것은 여러 가지 요소에 의해 영향을 받는다. 그러니, 탐색적 테스팅으로 작업을 진행해보고, 경험을 쌓아보라.

탐색적 테스팅은 마음 속의 무술로 묘사되기도 한다. 이것은 덤불로부터 튀어나오는 제품을 어떻게 다루고 테스팅이라는 결투에 어떻게 대응할 지의 문제다. 자, 당신은 이 책을 읽는다고 해서 검은 띠가 되는 것은 아니다. 당신은 이것을 직접 실행해야만 한다. ***Happy Practicing!***