정적 분석기를 믿고 따라와

박민솔

정적 분석기는 신뢰와 과감함의 균형이 중요하다. 먼저, 개발자들이 정적 분석기를 믿고 쓸 수 있어야 한다. 거짓 양성과 거짓 음성을 줄여서 정적 분석기가 최대한 정확한 판단을 할 수 있도록 해야 한다. 이와 동시에 개발자에게 오류를 고치는 방법을 제안하거나 개발자의 개발 환경 속에 밀착함으로써 좀 더 과감하게 개발자를 도와줄 수 있어야 한다. 특히 코드 작성 단계에서는 더욱 큰 과감함을 발휘할 기회가 주어진다. 연구자들이 믿을 수 있는 과감한 정적 분석기를 만들어 산업계와 선순환을 이룰 수 있기를 기대한다.

개발자가 믿을 수 있는, 큰 도움이 되는 정적 분석기가 필요하다. 기업에 정적 분석기를 도입할 때를 보면 어려움을 겪는 경우도, 순탄하게 진행되는 경우도 존재한다. 도입 성공과 실패를 가르는 데에는 정적 분석기 자체의 성능 및 제공하는 기능도 중요하지만, 그와 동시에 개발자들이 그 분석기를 믿고 쓸 수 있는지도 중요한 요인으로 작용했다. 성공한 사례들을 보면 대체로 개발자들의 신뢰를 얻은 뒤에는 정적 분석기에 더욱 다양한 기능을 추가하면서 영향력을 키워나갔다. 반대로 실패한 사례에서는 정적 분석기가 다양한 기능을 제공함에도 불구하고 개발자들이 잘 사용하지 않아서 어려움을 겪는 경우도 많았다. 즉, 정적 분석기는 신뢰와 과감함의 균형이 중요하다고 말할 수 있는 것이다.

개발자들이 정적 분석기를 신뢰하게 하려면 어떻게 해야 할까? 거짓 양성(false positive)이 많으면 개발자들은 정적 분석기를 신뢰하지 않는다. 오류가 있다고 해서 막상 들여다봤는데 오류가 아니라면 그 자체로 개발자를 방해하는 셈이 된다. 이러한 일이 잦아지면, 개발자들은 정적 분석기를 '말도 안 되는 소리를 하면서 자꾸 방해하는 잔소리꾼' 정도로 생각하게 된다. 장기적으로는 양치기 소년의 이야기처럼 더 이상 정적 분석기의 알림을 귀담아들으려고 하지 않을 것이다. 반대로 거짓 음성(false negative)이 많은 경우 개발자들은 정적 분석기를 통과한 프로그램을 신뢰하지 않는다. 오류가 없다고 판정된 프로그램이 사용자들에게 배포된 후에 오류가 발견된다면 이를 되돌리기 위해서는 매우 큰 비용이 발생한다. 이러한 일이 잦아지면, 앞으로 개발자들은 정적 분석기를 통과한 프로그램임에도 불구하고 여전히 오류가 있지 않을까 걱정하며 '보이지 않는 오류'와 싸워야 한다. 결국, 거짓 양성과 거짓 음성을 줄이고 정적 분석기가 최대한 정확한 판단을 하도록 만들어야 개발자들이 그 분석기를 신뢰할 수 있는 것이다.

거짓 양성과 거짓 음성을 최대한 줄임으로서 신뢰를 얻었다면, 정적 분석기는 조금 더 과감해질 필요가 있다. 과감하다는 것은, "내 분석 결과를 참고할 거면 참고해 " 느낌의 소극적인 태도를 보이기보다는, 개발자에게 당당하게 의견을 제시하거나 심지어는 정적 분석기를 사용하도록 강제하는 적극적인 태도를 뜻한다. 이러한 과감함을 통해, 정적 분석기는 본인을 믿고 사용해 주는 개발자들에게 더욱 많은 도움을 제공하도록 노력해야 한다. 틀릴 것을 각오하고, 오류를 고치는 방법(fix)까지 제시할 수 있어야 한다. 구글의 Tricorder 역시 오류를 고치는 방법을 제공하는데, 나는 이 기능이야말로 Tricorder가 성공할 수 있었던 큰 요인 중 하나라고 생각한다. 아무리 정적 분석기가 오류를 정확하게 짚어줘도, 그것을 해결하는 방법을 떠올리기는 쉽지 않을 수 있다. 물론 단순히 오타로 인해 발생한 간단한 오류도 있겠지만, 정적 분석기가 제공하는 오류 발생 경로를 따라가 보면서 한참 고민해 봐야 고칠 수 있는 오류도 있을 것이다. 만약 정적 분석기가 오류를

고치는 방법까지 제시한다면, 오류를 없애는 데 필요한 시간을 줄일 수 있으며 이는 개발자의 생산성을 크게 향상시킬 것이다. 설령 정적 분석기가 제시한 고치는 방법이 정확하지는 않더라도, 개발자가 이를 참고하여 비슷한 맥락으로 고칠 수 있기 때문에 시간이 단축되는 효과를 볼 수 있다. 심리적인 면에서도, 개발자는 본인이 정적 분석기에게 도움받고 있다는 것을 크게 체감할 수 있기 때문에 정적 분석기와의 신뢰 형성에 더욱 도움이 될 것이다.

정적 분석기가 개발자의 개발 환경으로 최대한 밀착하는 것 역시 좋은 과감함의 예시이다. 코드를 작성 중일 때부터 시작해서, 컴파일할 때, 코드 리뷰(code review)를 할 때, 저장소에 올라갔을 때 등 개발 과정 속 다양한 시점에서 정적 분석기가 활약할 수 있다. 이 중 어느 시점 하나 소홀히 하지 말고 늘 개발자의 곁에서 도움을 줄 수 있어야 한다. 물론 개발 과정 속 어느 부분에 밀착할지는 오류의 빈도나 심각성에 따라서 잘 결정해야 할 일이다. 잘못 밀착하는 경우 오히려 개발자를 귀찮게 하여 신뢰를 잃을 수도 있기 때문에 조심해야 한다.

과감함의 측면에서, 개발 과정 중 특히 코드를 작성 중인 단계에 주목하고 싶다. 개발자와 말 그대로 가장 가까운 단계이기 때문에, 더욱 많은 과감함을 발휘할 수 있기 때문이다. 현재에도 많은 개발 환경에서 작성 중인 코드의 문법 오류나 타입 검사를 수행하고 있다. 코드를 작성하고 있을 때 틀린 부분에 빨간 줄이 떠서 오류를 알아차린 경험은 코딩해 본 사람이라면 누구나 해봤을 것이다. 더 나아가서 정수 오버플로우(integer overflow) 등의 생각하기 까다로운 오류나 버퍼 오버플로우(buffer overflow) 등 심각한 보안 오류를 작성 중인 코드에서 탐지할 수 있다면, 오류 있는 코드가 배포되는 것을 가장 선제적으로 방어할 수 있으며 오류를 고치는 데 필요한 시간마저 절약할 수 있을 것이다. 오류 탐지 말고도 정적 분석을 통해 얻은 결과로 더 정확한 코드 완성(code completion) 기능을 제공하거나 더 정확하게 변수 이름을 추천해 주는 등 다양한 과감함을 발휘할 수 있는 곳이 바로 코드를 작성 중인 단계이다. 물론 그와 동시에 잘못된 오류 탐지, 잘못된 기능 제공으로 개발자를 타자 칠 때부터 귀찮게 만들지 않도록 더욱 조심해야 한다.

이렇게 믿을 수 있는 과감한 정적 분석기를 만드는 것은 연구자의 몫이다. 연구자는 개발자가 어느 시점에 어떤 분석 기능을 사용할지 잘 생각하면서 품질 좋은 정적 분석기를 만들어야 한다. 그렇게 된다면 많은 개발자가 그 분석기를 믿고 따르면서 도움을 받을 수 있으며, 거꾸로 실제 코드에서의 분석 데이터, 피드백(feedback) 및 오류 수정 제안 등으로 연구자에게 도움을 줄 수도 있을 것이다. 학계와 산업계의 선순환을 기대한다.