개발자의 마음을 움직이는 공학

이재현

좋은 글과 마찬가지로, 좋은 프로그램은 문법적으로 올바르고, 코딩 규칙이 통일되고, 간결하면서, 의도를 정확히 구현해야 한다. 그러한 좋은 프로그램을 만들기 위해서 보통 코드의 성질을 검사하고 퇴고하는 코드 리뷰 단계를 거친다. 복잡한 프로그램을 다루는 경우는 프로그램 분석기를 활용해 코드를 체계적으로 검사하고 리뷰한다. 그러나 분석 결과를 읽고 제대로 이해하여 실제 코드에 반영하는 것은 개발자에게 어렵고 귀찮은 일이다. 그렇기에 어떻게 분석을 더 잘할까 하는 기술적인 고민만큼이나, 어떻게 분석 결과를 더 설득력 있게 전달할까 하는 고민이 중요하다. 구글, 메타, 애플은 분석의 설득력을 높이기 위하여 분석기에게 근거를 들어 친절하게 설명하고, 빠르게 반응하며, 때로는 호되게 코드 수정을 요구하는 등의 성격을 부여한다. 이러한 노력 덕분에 엄청난 복잡성에도 불구하고 유지보수가 쉽고, 신뢰할 수 있는 훌륭한 프로그램을 만들 수 있었다. 결국 좋은 프로그램을 완성하는 것은 뛰어난 개발 실력과 분석 기술을 연결하는, 개발자의 마음을 움직이는 공학인 것이다.

좋은 프로그램을 만드는 것은 좋은 글을 쓰는 것과 닮아 있다. 좋은 글의 가장 우선적인 기준은 문법의 정확성이다. 프로그램도 마찬가지로, 최소한의 요구 사항은 문법(syntax)적 올바름이다. 그러면 문법이 올바른 프로그램은 그 자체로 좋은 프로그램일까? 물론 아니다. 문장 하나하나의 문법이 정확하더라도, 글 전체를 읽어 보면 맥락이 매끄럽지 않은 경우가 있다. 이와 유사하게, 프로그램도 맥락, 즉 의미 구조(semantics)를 따져야 한다. 좋은 프로그램은 통일성 있어야 한다. 협업하는 사람들 간에 같은 코딩 규칙(convention)을 사용하여 읽고 유지보수하기 쉬운 코드를 써야 한다. 비슷한 이유로, 중언부언을 피해 간결하고 군더더기 없는 코드를 작성해야 한다. 그리고 가장 중요한 것은, 여느 훌륭한 글이 그렇듯, 의도와 다르거나 틀린 내용을 써서는 안된다는 것이다.

글을 출판하기 앞서 퇴고를 꼼꼼히 하듯이, 개발자는 작성한 코드를 배포하기 전에 코드 리뷰라는 과정을 거친다. 새롭게 구현하거나, 기존의 논리를 수정한 프로그램이 앞서 이야기한 좋은 프로그램의 성질을 갖추고 있는지 검사하는 단계다. 프로그램은 문장(statement), 함수 등의 작은 요소들이 논리적으로 연결되어 서로 상호작용하는 방식으로 임무를 수행한다. 따라서 전체 코드의 아주 작은 부분을 수정했다 하더라도, 프로그램 전체에 생각지도 못한 결함을 초래할 수 있다. 2014년 발견된 애플의 goto fail 취약점이 좋은 예시다. 문제가 된 코드는 if (...) goto fail; goto fail; 이었다. goto fail; 을 실수로 하나 더 쓴 바람에 조건문의 결과와 상관없이 항상 fail로 이동하는 결함이 생겼다. 조건문에 항상 중괄호를 붙여 주는 코딩 규칙을 사용했더라면 쉽게 발견하고 예방할 수 있었을 결함이다. 이러한 실수를 최대한 예방하고 프로그램의 품질을 관리하기 위해 코드 리뷰가 꼭 필요하다.

효과적인 코드 리뷰를 위해 먼저 필요한 것은 코드 전체를 이해하는 주체다. 전체 코드의 동작 방식과 조직의 코딩 규칙을 모두 알아야 새로운 코드에 대해 적절히 평가할 수 있다. 보통은 팀장이나 선임 개발자가 이 역할을 맡는다. 하지만, 아무리 경험 많은 사람이라 하더라도 정확하고 빠짐 없는 리뷰를 하는 것은 쉽지 않다. 나 스스로도 아직 새내기 개발자이지만, 연구실에서 진행하는 프로젝트의 코드 리뷰를 맡은 경험이 있다. 코드를 찬찬히 읽고 LGTM(Looks good to me)이라는 리뷰를 남겼지만, 미처 예상하지 못한 오류가 있었던 것은 아닌지 걱정이 들었다. 내 경험과 비슷한 내용을 다룬 <how we write/review code in big tech companies>라는 유튜브 영상도 있다. 1000만 이상의 조회수를 기록한 것으로 보아, 사람이 작성한 코드 리뷰가 약간 미심쩍다는 공감대가 존재한다는 것을 알 수 있다. 몇만, 몇백만 줄의 복잡한 프로그램을 다루는 경우는 사람에 의존해 코드 리뷰를 하는 것이 사실상 불가능하다.

따라서, 큰 프로그램을 다루는 테크 회사들은 프로그램 분석기를 활용해 코드를 체계적으로 검사하고 리뷰한다. 프로그램 분석기는 프로그램을 입력으로 받아 그 성질을 분석하고 결함을 탐지하는 프로그램이다. 구글은 전사에 걸쳐 Tricorder라는 프로그램 분석기를 사용해 코딩 규칙의 통일성과 간단한 오류를 검사한다. 메타(구 페이스북)는 Infer라는 프로그램 정적 분석기를 사용해 페이스북, 인스타그램 등 어플리케이션의 결함을 찾아낸다. 이 정적 분석기는 요약 해석

(abstract interpretation)이라는, 프로그램을 실제로 실행하지 않고 가능한 다양한 실행 흐름과 결과를 예상하는 방식을 사용한다. 물론 프로그램 분석기가 항상 옳은 말만 하는 것은 아니다. 오류가 실제로는 없는데 있다고, 반대로 오류가 실제로는 있는데 없다고 오탐하기도 한다. 그렇지만 적어도 분석기는 사람보다 체계적으로, 빠르게, 거의 빠짐없이 프로그램을 검사할 수 있다.

분석기로 프로그램의 문제점을 찾았다고 끝이 아니다. "구슬이 서 말이라도 꿰어야 보배다"라는 속담처럼, 코드 리뷰를 통해 좋은 코드를 완성하는 것은 결국 개발자의 역할이다. 분석 결과를 곧이곧대로 전달만 하면 개발자가 그것을 전부 이해하여 코드에 반영할 것이라고 기대해서는 안 된다. 분석 결과가 부정확한 경우도 있기 때문에, 리뷰가 정말 맞는 말을 하는지 개발자가 스스로 따져 보는 수고가 필요하다. 또, 개발자는 리뷰의 우선순위를 매겨야 한다. 예를 들어, 작은 결함을 고치기 위해 전체 코드의 구조를 바꾸는 일에 시간을 쏟기보다는 프로그램의 안정성, 사용성과 관련된 리뷰를 먼저 처리하는 것이 효과적이다. 이처럼 리뷰를 이해하고 행동으로 옮기는 것은 어렵고 귀찮은 일이기 때문에 "분석기가 이렇게 말하더라" 식의 리뷰는 개발자의 마음을 움직이기에 충분하지 않다. 구글과 메타 모두 프로그램 분석기를 처음 도입했을 때, 분석 결과를 제대로 전달하지 못해 개발자들이 대부분을 무시하는 실패를 겪었다.

같은 내용의 리뷰라도 어떻게 전달하느냐에 따라 개발자의 행동이 달라지기에, 테크 회사에서는 분석기가 만든 리뷰의 설득력을 높이려 노력한다. 먼저, 개발자가 리뷰를 납득할 만한 근거를 친절히 제시하는 방법이 있다. 애플이 제공하는 Swift 프로그램 분석기는, 코드 위에 화살표를 그려서 어떤 경로를 통해 취약점이 발생할 수 있는지 시각적으로 보여준다. 또, 개발자가 즉각적으로 반응할 수 있도록 빠른 리뷰를 제공하려는 노력도 있다. 메타는 Infer 분석기를 조립식으로 설계하여 분석 결과를 얻기 위해 매번 전체 코드를 훑어볼 필요 없이, 변경된 코드에 관련한 작은 부분만 분석을 수행하도록 했다. 덕분에 개발자가 수정한 코드를 원격 저장소에 올리는 변경 시점(diff-time)마다 15분 이내로 분석 결과를 제공할 수 있다. 전체 코드를 분석하는 데에는 1시간 이상이 걸린다는 것을 생각하면, 이는 상당히 빠른 시간이다. 마지막으로, 분석 결과에 약간의 강제성을 더하는 방법도 있다. 모든 결과를 청유형으로 전달하면 "안해도 그만"이라는 인상을 줄 수 있으므로, 명령형을 적절히 섞는 것이다. 구글은 중요한 코딩 규칙에 대해서는 컴파일 경고(warning) 대신 에러를 발생시켜서 개발자들이 규칙을 지키도록 강제한다.

Copilot, ChatGPT 등의 거대 언어 모델로 코드를 생성할 미래에는 어떨까? 설득력 있는 분석기가 오히려 더 중요해질 것이라 예상한다. 거대 언어 모델이 나날이 발전하고 있지만, 그 기본 원리는 "그럴듯한 코드 생성"이기 때문에 규칙에 어긋나거나 취약점이 있는 코드가 만들어질 가능성을 배제할 수 없다. 그렇기에 최종적으로 인공지능이 만든 코드를 전체 코드에 반영할지 말지 결정하는 것은 여전히 인간 개발자의 몫일 것이다. 어찌 보면 개발자의 어깨가 더 무거워졌다고 볼 수 있다. 자신이 직접 작성하지도 않은 코드에 대해서 리뷰를 읽고, 이해하고, 적절한 조치를 취해야 하기 때문이다. 오류가 왜 발생할 수 있는지, 어떤 문제가 더 빨리 해결되어야 하는지 등의 설명을 충분히 제시하지 못하면 코드 리뷰 단계와 분석 기술이 있음에도 읽기 어렵고 틀린 코드가 세상에 나올 수 있다. 분석 결과에 설득력을 부여하는 기술이 더욱 중요해지는 셈이다.

정리하면, 좋은 프로그램을 만드는 것은 단순 기술의 영역을 넘어선 설득과 이해의 영역이다. 프로그램 분석 결과를 어떻게 전달하느냐에 따라서 개발자는 분석기를 이상한 소리만 늘어놓는 잔소리꾼으로 생각할 수도 있고, 프로그램을 올바른 방향으로 인도하는 조언가로 생각할 수도 있다. 조언가같은 분석기를 만들기 위해서는 빠르게 반응하고, 근거를 통해 부드럽게 설명하며, 때로는 호되게 이야기하는 등의 성격을 분석기 설계에 녹여 내는 치열한 고민이 있어야 한다. 훌륭한 프로그램이라 하면 문법, 의미 구조, 규칙, 오류, 정확성, 분석 등의 엄밀하고 딱딱한 단어가 먼저 떠오른다. 하지만 그 이면에는 결국 개발자의 마음을 움직이는 공학이 있다.