제 8 장 정적 테스트

8.1 개 요

소프트웨어 테스트 방법은 프로그램 실행을 요구하는 동적 테스트와 프로그램 실행을 요구하지 않는 정적 테스트로 분류한다. IEEE 1028-2008에서는 정적 테스트를 리뷰(Review)라고 하며 다음과 같이 감사를 포함하여 5종류로 분류한다.

- 관리 리뷰(Management review)
- 기술 리뷰(Technical review)
- 인스펙션(Inspection)
- 워크쓰루(Walk-through)
- 감사(Audit)

리뷰는 여러 전문가가 모여 프로그램을 검토하여 결함을 검출하는 방법이다. 리뷰 대상이 되는 작업물은 프로그램에 한정되지 않고 소프트웨어 개발 중에 생성되는 모든 산출물이다. 즉, 요구사항 명세서, 설계 명세서, 테스트 계획서와 같은 문서들이 올바르게 작성되었는지를 판단하기 위해 리뷰 방법을 이용할 수 있다. 테스트 대상으로 모든 산출물이 가능하다는 점은 결함 제거 비용 측면에서 매우 중요한 의미가 있다.

프로그램 개발이 완료된 후에 결함을 제거하는 일은 요구 단계나 설계 단계보다 100배 이상의 노력과 비용이 소요된다. 따라서 결함이 발생하는 순간 가능한 한 빨리 검출하여 제거해야 결함 제거 비용을 절약할 수 있다. 만약 결함이 발생하였는데 제거되지 않고 다음 단계로 전이될 경우 결함 제거 비용이 매우 증가하게 된다.

Tom Gilb은 리뷰를 이용하면 결함을 발견하고 제거하는 평균 비용을 60~80% 감소할 수 있다고 말한다. 이는 전통적인 방법에 따른 소프트웨어 테스트 노력의 40%만으로도 테스트할 수 있고, 소프트웨어 개발 각 단계에서 다음 단계로 전이되는 결함의 50~70%를 감소시키는 효과가 있다는 것이다.

8.2 리뷰 프로세스

IEEE 1028-2008에서는 1970년대 초 IBM에서 Fagan이 정립한 인스펙션 수행 프로세스에 기반을 둔 리뷰에 필요한 공통 프로세스를 제공한다. 프로세스를 구성하는 활동의 적용 수준은 리뷰의 종류에 따라 달라진다. 다음은 IEEE 1028-2008에서 정의한 리뷰 프로세스이다.

① **경영진 준비**(Management preparation): 경영진은 리뷰를 성공적으로 수행하기 위해 필요한 자원(스태프, 설비, 재원, 훈련 및 교육)을 제공하고 법규, 표준 및 관련 정책의 요구에 따른 리뷰 수행을 보장해야 한다.

② **리뷰 계획**(Planning review): 리뷰 리더(Review leader)는 리뷰 목적을 파악해서 리뷰 팀을 구성하고 각 팀 구성원에게 책임을 할당하며 리뷰 참가자들에게 리뷰에 필요한 자료들을 제공한다. 또한, 리뷰 일정을 결정해서 참가자들에게 공지한다.

③ **리뷰 절차 개요 설명**(Overview of review procedures): 리뷰 리더의 요청이 있을 때 수행되며, 리뷰 목적과 리뷰 절차를 설명한다. 반드시 리뷰 리더가 할 필요는 없고 적절한 자격이 있는 사람이 이 단계를 수행해도 무방하다.

④ **작업물 개요 설명**(Overview of software product): 리뷰 리더의 요청이 있을 때 수행된다. 이는 리뷰 참가자들이 검토할 작업물을 좀 더 친숙하게 느끼게 하고 사전 이해도를 높이기 위함이다.

⑤ **개별 준비**(Individual preparation): 그룹 검토에 앞서 리뷰팀 멤버들은 개별적으로 작업물(Software product)이나 프로세스를 검토한다. 만약 검토 수행 중에 문제(Anomaly)를 발견하면 발견한 문제를 문서화하여 리뷰 리더에게 보낸다.

⑥ **그룹 검토**(Group examination): 계획한 날짜에 모든 리뷰 멤버들이 참가하는 회의를 한다. 여기에서 모든 검토 결과를 모아 검토 대상이 되는 작업물(또는 프로세스)의 상태에 관해 합의를 도출한다.

⑦ **재작업**(Rework): 검출된 문제 목록을 개발자나 문제 해결 책임자에게 전달하여 문제를 해결하는 작업을 수행한다.

⑧ **후속작업**(Follow up): 리뷰 리더는 회의에서 산출된 모든 조치 항목(Action item)을 완료(Close)하였는지 확인한다.

이러한 리뷰 프로세스를 수행하기 전에 리뷰 프로세스를 수행하는 데 필요한 조건(Entry condition)들이 만족되었는지 확인할 필요가 있으며, 리뷰가 끝난 후에도 성공적인 리뷰에 필요한 모든 행위가 수행되었는지 확인하고 리뷰 종류에 따라 적절한 리뷰 결과물이 산출되었는지 점검하는 과정이 필요하다.

8.3 관리 리뷰

관리 리뷰(Management review)의 목적은 진행 상황을 모니터하고 계획과 현재 일정 상태를 평가하여 필요하다면 자원, 일정이나 프로젝트 범위 등을 변경하는 것이다. 따라서 관리 리뷰 후에는 해당 계획이 적절히 변경되었는지 확인할 필요가 있다. 주로 관리 리뷰 대상이 되는 작업물은 다음과 같이 계획과 관련되거나 현재 프로젝트 진행 상황을 파악할 수 있는 관련 문서들이다.

- 설치 계획
- 백업 및 회복 계획
- 안정성 계획
- 재난 계획
- 비상 대책 계획(Contingency plan)
- 진행 보고서
- 테스트 결과

관리 리뷰는 8.2.절에 기술한 리뷰 프로세스를 따르지만 ④ 작업물 개요 설명 단계는 수행되지 않는다. 그룹 검토 회의는 관리 스태프들이 참여하며 관리자가 주재한다. 또한, 기술 스태프는 관리 스태프에게 필요한 정보를 제공해야 한다. 리뷰팀이 발견한 문제, 조치 사항, 결정사항 및 추천들은 서기(Recorder)가 기록한다. 조치 항목 목록, 문제(Anomaly list) 등은 의사 결정자(Decision maker)에게 전달되고 리뷰 목적이 달성되었는지 판단한다. 또한, 산출물들은 검토 회의 결과에 영향을 받는 사람들에게도 전달되어야 한다.

8.4 기술 리뷰

기술 리뷰(Technical review)는 유능한 인력으로 구성된 팀이 다음과 같은 작업을 수행하여 프로젝트의 기술적 상태를 확인하는 증거로 관리자에게 제공한다:

- 대상 작업물이 의도된 사용에 적합한지(Fitness for its intended use) 평가한다.
- 대상 작업물이 계획, 법규, 표준이나 명세를 충실히 지키는지 평가한다.
- 변경 사항이 적절하게 구현되었는지를 평가하고 변경 명세(Change specification)에 식별된 영역에만 해당 변경이 영향을 미치는지 평가한다.
- 여러 대안을 추천하거나 대안들을 검토한다.

기술 리뷰의 대상에는 소프트웨어 요구·설계 명세, 테스트 문서, 유지 보수 매뉴얼이나 설치 절차 등이 포함되지만 이에 한정되지는 않는다. 기술 리뷰는 기본적으로 8.2절에 기술한 리뷰 프로세스를 따른다. 기술 리뷰는 대표 엔지니어(Lead engineer)가 주재하며 경우에 따라 관리자가 해결해야 할 이슈가 있으면 관리자도 참가할 수 있다.

8.5 인스펙션

인스펙션(Inspection)은 IEEE 1028-2008의 리뷰 종류 중에서 가장 형식화된 대표적인 리뷰 방식이다. IBM의 Fagan은 결함은 발생한 시점에서 가능한 한 이른 시기에 제거해야 하며, 이를 위해서는 더욱 구조적이고 절차적인 방법이 필요하다는 생각을 시작으로 인스펙션 프로세스를 정립하였다.

인스펙션은 동료 검토(Peer review)라고 할 수 있다. 동료 검토란 비슷한 수준이나 역할을 가진 사람들이 코드 등을 포함한 소프트웨어 산출물을 검토하는 작업이다. 인스펙션은 가능한 한 개발 초기에 검사해야만 개발 초기 작업물에서 문제를 찾아낼 수 있다. 인스펙션으로 검출한 문제(Anomaly)의 유형이나 투자한 시간 등을 포함한 자료를 수집하여 인스펙션 자체 프로세스와 체크리스트 같은 인스펙션 지원 문서 개선에 활용한다.

8.5.1 인스펙션 참가자의 역할

인스펙션 과정을 이해하면 인스펙션을 더욱 효율적으로 수행하여 자원과 시간의 낭비를 줄일 수 있다. 인스펙션 과정을 올바르게 이해하기 위해 인스펙션 참가자의 역할을 알아보는 것은 매우 중요하다.

- **주재자(Inspection leader/Moderator)**: 인스펙션 주재자의 주된 임무는 검사할 작업물을 기초로 인스펙션 참가자들을 선정하고 인스펙션을 계획하는 일이다. 인스펙션 회의를 위해 주재자는 인스펙션 참가자들에게 미리 검토할 자료를 전달하여 참가자들이 인스펙션 회의를 충분히 준비할 수 있게 해야 한다. 인스펙션 회의에서 주재자는 회의를 주재하는 역할을 담당한다. 회의나 준비 기간 동안 발생할 수 있는 참가자 간의 불화 또는 예기치 않은 상황에 잘 대처하고, 회의가 끝난 뒤에는 어떤 후속 조치가 필요한지 신속히 결정해야 한다. 또한, 회의 후에는 회의에서 기록된 모든 자료를 보고서 형식으로 만들어 소프트웨어 개발자에게 전달하고 인스펙션 프로세스를 개선하기 위해 사용되는 여러 데이터를 수집한다. 인스펙션 주재자는 전문적으로 훈련된 퍼실리테이터(Facilitator)가 담당한다(심화노트 참조).

- **작성자(Author)**: 작성자는 인스펙션 회의에 필요한 자료를 제출해야 하며 자료 내용에 관한 설명을 하거나 질문에 대답할 수 있어야 한다. 회의 내용을 신중히 듣고 토의에 참여해야 하며 방어적인 자세로 회의에 참여해서는 안 된다. 회의가 끝난 후에는 검출된 결함 내용에 관하여 재작업을 해야 한다. 8.2.절의 리뷰 프로세스 중 단계 ④ 작업물 개요 설명도 작성자가 수행한다. IEEE 1028-2008에서 작성자는 인스펙션 주재자가 될 수 없다.

- **낭독자(Reader)**: 낭독자는 작업물에 대한 자신의 이해 및 해석을 바탕으로 작업물에 대해 회의 참가자들에게 설명하며 인스펙션 회의를 이끄는 역할을 수행한다. 준비에 필요한 시간을 줄이기 위하여 작업물을 여러 부분으로 분할하고 각 분할된 부분에 다른 낭독자들을 할당하여 회의를 진행할 수도 있다.

- **기록자(Recorder)**: 기록자는 인스펙션 회의에서 논쟁, 모든 질문 및 답변 등을 기록해야 한다. 또한, 인스펙션 회의를 마치면 기록된 사항들을 정리하여 문서화해야 한다. 기록자는 회의에서 회의 주재자나 개발자 입장이 아닌 검토자 입장에서 참가해야 한다. 인스펙션 회의의 토론을 이해하고 의미 있는 문서를 위해 충분한 지식을 갖추어야 한다. IEEE 1028-2008에서 작성자는 기록자가 될 수 없지만, 인스펙션 주재자는 기록자 역할도 수행할 수 있는 것으로 명시하고 있다.

- 검토자(Inspector): 검토자는 인스펙션 회의를 위해서 전달받은 자료를 충분히 검토하고 인스펙션 회의를 준비해야 한다. 또한, 검토할 작업물을 충분히 이해하도록 노력해야 하고 주어진 자료에서 결함을 찾아내고 기록한다. 몇몇 검토자에게는 특별한 주제를 할당하는 것도 좋다. 예를 들면, 어떤 검토자는 특정 표준에 부합되는지에 중점을 두어 리뷰하고, 어떤 검토자는 철자 결함 등에 중점을 두어 리뷰를 수행할 수 있다. 또한, 검토자는 해결자 입장이 아니므로 찾아낸 결함을 해결하기 위해 노력하지 말고 간단히 의견만 제시해야 한다. 다른 사람의 의견이나 작성자를 비판하기 위해 인스펙션 회의에 참여하는 것이 아니라 인스펙션 팀을 돕기 위해 참가하는 것이므로 개인적인 친분은 배제하고 최대한 객관적인 관점에서 회의에 참여해야 한다. 또한, 관리자 직책을 담당하는 사람은 팀 멤버로 참여하는 것이 금지된다.

심화노트 **퍼실리테이터**

퍼실리테이터(Facilitator)는 검토 회의 또는 워크숍과 같이 여러 사람이 일정한 목적을 가지고 함께 작업을 수행할 때, 효과적으로 그 목적을 달성하도록 작업 과정을 설계하고 참여를 유도하여 질 높은 결과물이 나오도록 도움을 주는 사람을 말한다.

이상적인 인스펙션 팀의 규모는 어떻게 될까? 작성자가 자신의 작업물을 비평하고 검토할 수는 없기 때문에 작성자 이외에 검토자가 최소한 한 명은 있어야 하고, 회의를 주재하는 사람이 필요하므로 인스펙션 팀은 두 명 이상이어야 한다. 400개의 인스펙션에서 자료를 수집한 결과에 따르면 4명의 팀이 가장 이상적인 규모라고 하며 일반적으로 3~6명의 규모로 팀이 구성된다. 6명 이상인 팀은 관리하기가 힘들기 때문에 좋지 않다. 또한, 주재자와 같은 다른 역할이 이미 있다 할지라도 팀 구성원 모두는 검토자 역할을 수행한다.

팀을 구성할 때 중요한 점은 절대 작성자의 자질을 평가하는 분위기가 형성되어서는 안 된다는 점이다. 성공적인 인스펙션이 되기 위해서는 인스펙션의 목적이 작성자의 능력을 평가하는 것이 아니라 작업물에서 결함을 발견하여 더 나은 품질의 소프트웨어를 개발하는 데 있다는 점을 팀 구성원 모두가 인식해야 한다.

8.5.2 인스펙션 과정

인스펙션은 기본적으로 8.2.절에서 기술한 리뷰 프로세스를 따른다. 인스펙션과 각 단계에 관한 설명은 다음과 같다.

① **리뷰 계획**(Planning review): 리뷰 리더(Review leader)인 중재자가 리뷰 목적을 파악해서 리뷰팀을 구성한 후, 각 팀 구성원에게 책임을 할당하고 리뷰 참가자들에게 리뷰 날짜, 장소 등을 알린다. 또한, 인스펙션에 필요한 자료들을 참가자들에게 제공하고 준비에 필요한 시간을 할애한다.

② **인스펙션 절차 개요 설명**(Overview of inspection procedures): 중재자는 참가자들의 역할을 할당하고 체크리스트나 역할 할당에 관한 질문에 대답한다. 또한, 중재자는 참가자에게 최소 준비 시간, 희망하는 인스펙션 수행률(Inspection rate) 및 유사한 프로젝트의 인스펙션에서 검출된 문제의 수 등을 전달한다.

③ **인스펙션 작업물에 대한 개요 설명**(Overview of inspection product): 작성자가 검토자들에게 검토할 작업물(예를 들면, 코드 인스펙션인 경우에는 프로그램 코드)에 대해 설명한다. 이는 검토할 작업물을 좀 더 친숙하게 느끼게 하고 사전 이해도를 높이기 위함이다. 검토자는 작업물에 대한 이해도를 높이기 위한 질문을 할 수 있지만, 이 단계에서 더 나은 해결책(설계 또는 구현 방법)을 제안하거나 결함을 수정하는 것과 같은 작업은 하지 않는다.

④ **준비**(Preparation): 실제 리뷰 회의 전에 인스펙션팀 구성원은 작업물을 검토한다. 준비 기간에 검출된 문제들은 중재자에게 전달한다. 중재자는 검출된 문제들을 IEEE 1044 등을 활용해 적절하게 분류하여 인스펙션을 계속할지 판단한다. 만약 대상 작업물의 문제들이 인스펙션을 수행하지 못할 정도로 심각하거나 많다면 인스펙션을 취소할 수도 있다. 중재자는 검출한 문제 목록을 작성자에게 전달한다.

⑤ **검토 회의**(Inspection meeting): 개별적으로 체크리스트를 사용하여 작업물에 대한 개별 검토가 완료된 후, 모든 검토자가 참가하는 회의를 시작한다. 회의에서는 낭독자가 인스펙션 참가자들에게 작업물에 대한 설명을 하며 참가자들은 객관적으로 철저하게 작업물을 검사한다. 기록자는 검출한 문제들을 분류하고 관련된 정보를 기록한다. 검토 회의에서 작성자의 역할은 자신의 작업물에 대한 검토자의 질문에 답하며 참가자들의 결함 검출에 도움을 주는 일이다. 또한, 검토자에게서 피드백을 받는 역할로 참가하며 작업물

에 대해 더 설명하거나 옹호하는 행동은 피해야 한다. 그 이유는 인스펙션의 목적에는 작업물 자체를 누가 보더라도 이해하기 쉽게 작성되었다는 것을 확인하는 작업도 포함되기 때문이다. IEEE 1028-2008에서는 검토 회의 결과를 다음과 같이 분류하여 규정한다.

- Accept with no verification or with rework verification: 있는 그대로 작업물을 승인하거나 약간의 재작업만을 수행한다.
- Accept with rework verification: 주재자나 주재자에게서 위임받은 사람이 재작업을 검증한 후에 작업물을 승인한다.
- Reinspect: 작업물을 승인할 수 없어 문제가 해결된 후에 재작업을 검증하기 위해 다시 인스펙션 일정을 계획한다.

⑥ **재작업**(Rework): 검출된 문제 목록이 작성자에게 전달되면 작성자는 실제 작업물에서 문제를 해결하는 작업을 수행한다. 예를 들면, 코드 인스펙션에서 이러한 작업에는 코드를 변경하는 작업, 주석문을 추가하거나 제거하는 작업 및 프로그램 구조를 재구성하는 작업이 포함될 수 있다. 이때 명심할 점은 검토 회의에서는 문제를 해결하는 방안에 관한 토론은 하지 않는다는 사실이다. 만약 작성자가 해결방안에 대한 피드백을 원한다면 검토 회의 후 작성자가 주체가 되어 검토자에게 문의할 수 있다.

⑦ **후속작업**(Follow up): 이 단계에서 주재자나 주재자에게서 위임받은 사람이 발견된 모든 문제에 대해 재작업이 충분하게 이루어지는지 확인한다.

8.6 워크쓰루

워크쓰루(Walkthroughs)는 인스펙션보다는 비형식적인 결함 검출 방법이다. 워크쓰루는 결함을 검출할 뿐만 아니라 참가자들의 교육이나 지식 공유를 위해 수행되기도 한다. 인스펙션에서 회의 주재자는 작성자가 아닌 사람이 맡지만, 워크쓰루는 작성자 본인이 보통 회의를 주재하며 기록자 역할도 담당할 수 있다. 인스펙션과 마찬가지로 관리자 직책을 담당하는 사람은 팀 멤버로 참여하는 것을 금지하고 있다.

IEEE 1024-2008에서는 워크쓰루 절차 및 작업물에 대한 개요 설명 단계를 워크쓰루 회의의 한 부분으로 다룰 수 있다고 명시해 놓았다. 워크쓰루를 행하는 방법은 단어가 의미하는 바와 같이 작성자가 작업물을 따라 돌아다니면서(Walkthroughs) 작업물에 대한 설명

을 진행하고 검출된 결함에 대한 권고 및 조치 사항들을 기록한다. 재작업 및 후속 단계에서 작성자는 모든 조치 사항들이 종결되었음을 확인한다.

8.7 감사

IEEE 1028-2008에서는 감사(Audit)의 목적을 소프트웨어 제품 및 프로세스가 규제, 표준, 가이드라인, 계획, 절차를 준수하고 있는지를 독립적으로 평가하는 것으로 규정하고 있다. 감사는 소프트웨어 제품의 제공자, 소비자, 또는 FDA와 같은 제3기관에서 필요에 따라 요구될 수 있다.

감사 참여자들은 특정 역할, 즉 대표 감사자(Lead auditor), 감사자(Auditor), 기록자(Recorder) 또는 개시자(Initiator) 등이 부여되며, 피감사 조직의 대표도 된다. 대표 감사자(Lead auditor)가 감사를 주도하며 인터뷰나 문서들을 점검함으로써 법규나 표준 등에 부합되는지 증거를 수집한다. 감사는 비준수 사항의 사례(Instance)를 식별하고 해당팀이 교정 활동(Corrective action)을 요구하는 보고서를 산출하게 한다.

8.8 정적 분석

지금까지 열거한 방법들은 기본적으로 사람이 직접 수행하는 수작업 중심의 방법이다. 이 방법들 외에도 자동화된 도구를 사용하여 정적 테스트를 수행할 수 있다. 예를 들면, 컴파일러는 문법 결함 등과 같은 기본적인 결함들도 검출할 수 있을 뿐만 아니라 경우에 따라 프로그램 내에 무한 루프나 절대 실행할 수 없는 프로그램 문장 등을 검출하는 기능도 수행한다. 이와 같이 도구의 지원을 받아 정적 테스트를 수행하는 것을 정적 분석(Static analysis)이라고 한다. 정적 분석으로 많은 작업을 수행할 수 있지만, 이 장에서는 코딩 표준 부합, 코드 복잡도 계산, 자료 흐름 분석을 소개한다.

8.8.1 코딩 표준

코딩 표준(Coding standard) 또는 코딩 지침(Coding guideline)은 개발자가 프로그램을 작성할 때 지켜야 하는 규약이다. 개발자는 자신이 선호하는 코딩 스타일이 있다. 예를 들

어, 주석문을 언제/어떻게 넣을 것인지, 변수와 함수 이름을 어떻게 지을 것인지, 들여쓰기를 어떻게 할 것인지와 같은 사항들이 코딩 스타일의 예이다. 또한, switch 문을 작성할 때 default 문을 넣을 것인지도 여기에 포함된다.

코딩 표준은 개발자가 자신의 스타일로 코딩을 하는 대신에 코딩 표준에 따라 일관되게 프로그램을 작성하게 하는 것이 목적이다. 즉, 한 프로그램을 여러 개발자가 협력하여 작성하더라도 프로그램 모든 부분이 동일한 개발자가 코딩한 것처럼 보이게 한다. 이는 가독성이 좋고, 이해하기 쉬우므로 유지보수가 용이한 프로그램을 개발할 수 있다.

예를 들어 그림 8.1의 프로그램을 보자.

```
int foo(int n) { int i = 0; while (n > 0) {i = i + n % 10;
n = n / 10;
} return i;}
```

그림 8.1 코딩 표준을 적용하기 전 프로그램

이 코드를 개발한 당사자도 한 달 후 아니 일주일 후에 코드를 보면 무슨 코드인지 이해하는 데 어려움이 따를 것이다.

그림 8.2의 코드는 위 코드를 구글(Google)의 자바(Java) 코딩 표준(심화노트 참고)을 따라 다시 작성한 것이다. 코딩 표준을 적용하기 전보다 가독성이 훨씬 좋아졌음을 알 수 있다.

```
public static int foo(int n) {
  int i = 0;
  while (n > 0) {
    i = i + n % 10;
    n = n/ 10;
  }
  return i;
}
```

그림 8.2 구글 자바 코딩 스타일에 따라 개선한 프로그램

심화노트 코딩 스타일

- 중괄호의 위치에 따른 코딩 스타일에는 대표적으로 BSD, K&R, GNU 3가지가 있다. 구글 자바 코딩 스타일은 K&R을 따른다.

BSD	K&R	GNU

`if (...)`
`{`
`  doSomething();`
`}`
	`if (...){`
	`  doSomething();`
	`}`
		`if(...)`
		`  {`
		`  doSomething();`
		`  }`

블록을 if 문 아래에 작성하고 들여쓰기를 하지만 중괄호는 들여쓰기를 하지 않는다.	블록의 여는 중괄호를 if와 같은 행에 배치한다.	블록을 if 문 아래에 작성하고 들여쓰기를 한다.

- 새로운 블록은 공백 두 칸의 들여쓰기로 시작한다. 블록이 끝나면 이전 들여쓰기 수준으로 되돌아간다.
- 메소드 이름은 카멜 표기 방식(Camel Case)을 따르고 동사나 동사구를 사용한다. 카멜 표기 방식이란 여러 단어를 연달아 사용할 때 각 단어의 첫 글자를 대문자로 적되, 맨 앞에 오는 글자는 소문자로 표기하는 것이다.
- 변수 이름은 메소드 이름과 같이 카멜 표기 방식을 따르지만, 명사를 사용한다. 또한, 의미가 있도록 작성되어야 하며 사용 의도를 드러내야 한다.

가독성은 개선되었지만, 여전히 프로그램을 이해하기가 어렵다. 그 이유는 함수 이름과 변수 이름들이 사용 의도를 전혀 나타내지 않기 때문이다. 그림 8.3의 코드는 함수 이름과 변수 이름을 사용 의도를 명확하게 드러내도록 심화노트의 마지막 두 항목에 따라 개선하였다. 실제 이 프로그램은 정수를 입력으로 받아 정수를 구성하는 숫자들의 합을 구하는 프로그램이다.

```
public static int getSumOfDigits(int n) {
  int sum = 0;
  while (n > 0) {
    sum = sum + n % 10;
    n = n/ 10;
  }
  return sum;
}
```

그림 8.3 함수 이름과 변수 이름을 개선한 프로그램

자동차, 우주/항공, 의료장비, 국방, 철도 등 안정성 및 신뢰성을 요구하는 분야에서도 코딩 가이드라인은 매우 중요한 역할을 한다. 이와 관련하여 대표적인 것이 MISRA-C이다.

MISRA-C는 영국 자동차 산업 신뢰성 협회(MISRA: Motor Industry Software Reliability Association)에서 발표한 C 프로그래밍 언어의 코딩 가이드라인이다. 이 코딩 가이드라인의 목적은 임베디드 시스템에서 안전하고, 이식성이 좋고, 신뢰성 있는 코드를 만드는 것이다.

MISRA-C는 1998년에 처음 등장했으며 이를 자동차 등 안전성이 요구되는 분야에 사용할 수 있도록 MISRA-C:2004라는 개정판이 발표되었다. 세 번째 버전인 MISRA-C:2012는 2013년 3월 발표되었다.

많은 언어가 개발되었지만, 지금까지도 여전히 C 언어를 임베디드 시스템을 개발할 때 선호한다. 그러나 C 언어는 안전성이 요구되는 시스템에서 사용될 때 많은 주의가 필요하다. 그 이유는 C 언어의 "undefined behavior"에 기인하여 문제가 발생하는 경우가 많기 때문이다. ISO/IEC 9899-2011과 MISRA-C에서는 "undefined behavior"를 다음과 같이 정의한다.

```
ISO/IEC 9899-2011: behavior, upon use of a nonportable or erroneous program
construct or of erroneous data, for which this International Standard imposes
no requirements. (호환이 안 되거나 잘못된 프로그램 구조나 데이터의 사용으로 인한 (프로
그램) 동작이며, 본 국제표준은 어떤 요건도 (컴파일러 개발자에게) 부과하지 않는다.)

MISRA-C: essentially programming errors, but for which the compiler writer
is not obliged to provide error messages. (컴파일러 개발자가 반드시 오류 메시지를
제공할 의무가 없는 프로그래밍 오류)
```

다음은 "undefined behavior"가 발생할 수 있는 경우이다.

- 초기화되지 않은 변수의 사용
- 배열의 범위를 넘어서는 인덱스를 사용하여 배열 참조
- 0으로 나눗셈 연산 수행

예를 들어, 그림 8.4 프로그램을 살펴보자.

```
int x;
if (x == 1) printf("Hello");
if (x == 2) printf("World!!!");
```

그림 8.4 "undefined behavior"가 발생하는 코드

이 프로그램은 정수형 변수 x를 초기화하지 않고 바로 사용하는 코드이다. 대부분의 사람은 이 코드를 보고 "Hello"나 "World!!!"를 출력할 것으로 생각한다. 그러나 이 경우 "undefined

behavior"가 발생된다. 즉, C 언어 명세에서는 이 코드에 대해 컴파일러가 어떤 목적 코드를 생성하도록 어떠한 요구도 하지 않는다. 따라서 "Hello"와 "World!!!"가 같이 출력되어도 하등 이상할 것이 없다. 더 나아가 현재 기억장치의 모든 파일을 지우는 행위를 수행할 수도 있는 것이다.

MISRA-C는 가급적 "undefined behavior"에 의존하지 않은 안전한 프로그램을 개발하는 것이 주요 목표이다. 예를 들어, MISRA-C:2004에는 그림 8.4와 같은 프로그램의 개발을 피하기 위해 다음과 같은 규칙 9.1이 있다.

```
Rule 9.1: All automatic variables shall have been assigned a value before
being used. (모든 자동 변수는 사용되기 전에 값을 할당받아야 한다.)
```

코딩 규칙을 수작업으로 검사하기란 매우 번거로운 일이다. 이를 자동화된 도구를 사용하면 많은 노력을 줄일 수 있다. 시중에 판매되는 몇몇 정적 분석 도구에서 MISRA-C 코딩 규칙을 검사할 수 있다. 대표적인 도구로는 LDRA, PRQA, Parasoft에서 판매하는 제품이 있다. 국내에서도 Suresoftetch 회사의 제품이 있다.

8.8.2 복잡도 분석

소프트웨어 개발의 핵심은 복잡도를 통제하여 필요 이상으로 복잡도가 높지 않도록 하는 일이다. 복잡도가 높은 프로그램은 신뢰성, 테스트 비용, 유지보수성 측면에서 좋지 않은 결과를 가져오기 때문이다. 프로그램 복잡도 통제를 위해 신뢰할 수 있는 척도를 사용하여 프로그램의 복잡도를 측정하는 것은 매우 중요한 작업이다.

지금까지 수많은 복잡도를 나타내는 지표가 있지만, 그중에서도 1976년에 McCabe가 발표한 순환 복잡도(Cyclomatic complexity)가 가장 널리 사용되고 있으며 많은 정적 도구가 이 지표를 지원하고 있다. 순환 복잡도는 주어진 제어 흐름 그래프(9.2절 참조)에서 선형적으로 독립적인 기본 경로(Basis path)라 불리는 프로그램 경로들의 개수이다. 이 기본 경로 집합을 사용하여 프로그램에 존재하는 어떤 경로도 선형적으로 조합하여 표현할 수 있는 특징을 지니고 있다. 순환 복잡도는 프로그램을 제어 흐름 그래프로 변환 후에 다음과 같이 구할 수 있다:

$$\text{순환 복잡도} = E - N + 2$$

여기에서 E는 제어 흐름 그래프에 있는 간선들의 개수이고, N은 노드들의 개수이다. 그림

8.5에서 E=8, N=7이므로 기본 경로들의 개수는 8−7+2=3개가 존재하며, 따라서 순환 복잡도도 3이다.

```
i = 1;
while (i<n) {
        if (i>10) d = d+10;
        else d = d-10;
        i++;
}
printf("%d", d);
```

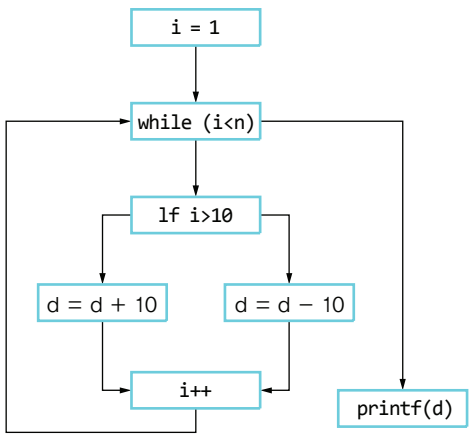


그림 8.5 예제 프로그램과 제어 흐름 그래프

또한, 순환 복잡도는 제어 흐름 그래프의 닫힌 영역(Region)들의 개수와 관계가 있다. 만약 제어 흐름 그래프 종료 노드부터 시작 노드로 간선을 연결한다면 제어 흐름 그래프가 간선과 노드들에 의해 둘러싸인 여러 개의 영역으로 분할된다. 순환 복잡도는 이와 같이 분할된 영역들의 개수이다. 그림 8.6은 그림 8.5에 주어진 제어 흐름 그래프의 영역을 보여주며 2개의 영역이 있음을 알 수 있다. 순한 복잡도를 닫힌 영역의 개수를 이용하여 계산하려면 다음의 공식을 사용한다:

순환 복잡도 = 닫힌 영역의 개수 + 1

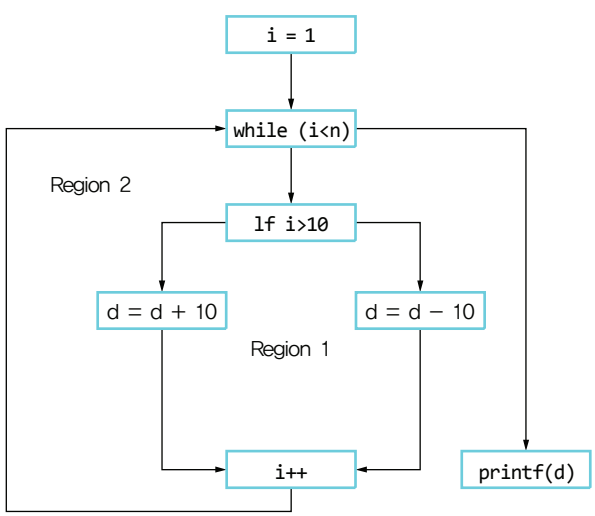


그림 8.6 제어 흐름 그래프에 존재하는 영역

더 간단하게 순환 복잡도를 계산하려면 제어 흐름 그래프에 있는 분기 노드들의 개수에 1을 더하면 된다.

즉,

순환 복잡도 = 분기 노드들의 개수 + 1

이 경우에도 그림 8.5에 주어진 프로그램의 순환 복잡도는 3임을 알 수 있다. McCabe에 따르면 순환 복잡도는 프로그램을 이해하고 테스트 및 유지보수가 얼마나 어려운지를 판별하는 기반이 될 수 있다.

Walsh는 1979년에 임베디드 시스템 모듈의 순환 복잡도가 10 이상일 때 결함률이 확연하게 증가한다는 연구 결과를 발표하였다. Audi, BMW, Porshe, 폭스바겐, 크라이슬러 연합, HIS에서도 10을 순환 복잡도의 제한 값으로 유지하도록 권고하였다. 반면에 JSF(Joint Strike Fighter) AV C++ 코딩 표준의 AV rule 3에서는 20 이하로 유지할 것을 권고하며 MISRA Report 5에서는 15를 넘어서는 안 된다고 기술하고 있다.

미국 카네기 멜론 대학교(Carnegie Mellon University)의 소프트웨어공학연구소(SEI: Software Engineering Institute)는 순환 복잡도에 따른 위험성(Risk) 증가를 언급하였고, RIAC(Reliability Information Analysis Center)는 복잡도와 소프트웨어 신뢰성과의 관계를 표 8.1과 같이 제시한다.

표 8.1 **순환 복잡도와 신뢰성과의 관계**

복잡도	신뢰성
10 이하	acceptable
30 이상	questionable
50 이상	cannot be tested
75 이상	bad fix (결함을 수정하기 위해 수행한 작업이 새로운 결함을 도입한다)

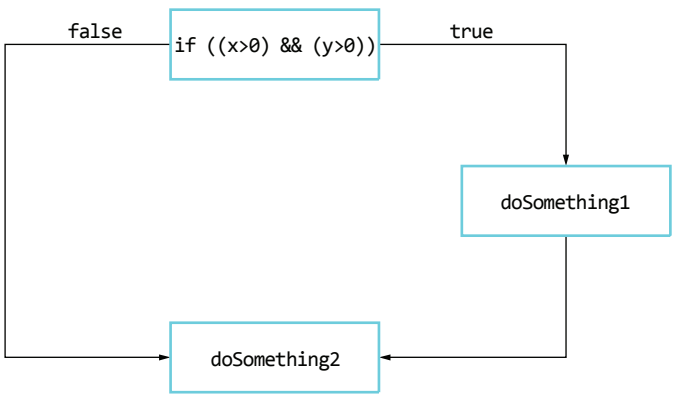
실확노트 순환 복잡도 계산

만약 분기 노드가 여러 개의 조건문으로 구성되어 있을 때 순환 복잡도를 어떻게 계산할까? 예를 들어, 다음 프로그램의 순환 복잡도를 계산해보자.

```
if ((x>0) && (y>0)) {
      doSomething1;
```

```
}
doSomething2;
```

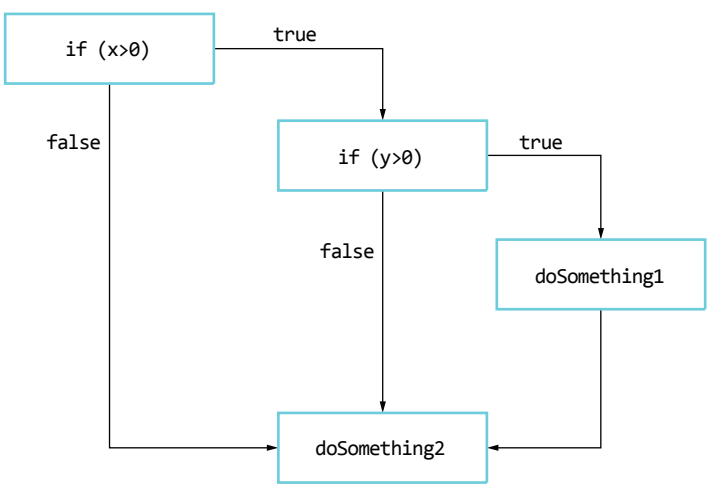
이 프로그램의 제어 흐름 그래프는 다음과 같다.



이 그래프에서 순환 복잡도를 산출해보자. E-N+2=3-3+2이므로 순환 복잡도는 2이다. 그러나 분기 노드가 복합 조건식이므로 이를 감안하여 계산할 필요가 있다. 또한, 단축연산을 수행할 경우에는 예제 프로그램은 아래와 같이 변환된다.

```
if (x>0) {
      if (y>0)
        doSomething1;
}
doSomething2;
```

이 프로그램에 해당하는 제어 흐름 그래프는 다음과 같다.



이 그래프에서 순환 복잡도를 산출해보자. E-N+2=5-4+2=3이므로 순환 복잡도는 3이다. 일반적인 순환 복잡도는 다음과 같이 계산한다.

- if, while, for, &&, ||를 만나면 +1을 한다.
- switch 문의 각 case에 +1을 한다.

switch 문 case마다 복잡도를 증가하면 프로그램 복잡도가 필요 이상으로 증가될 수 있다. 어떤 경우는 이를 반영하여 switch 문 자체로 +1을 증가하고 case는 고려하지 않는 경우도 있다.

8.8.3