제 14 장 테스트 실행 및 결함 보고

14.1 개 요

테스트 실행 활동은 테스트 설계 및 개발 활동에서 개발된 테스트 절차들을 실행하여 실행 결과를 테스트 실행 로그에 기록한다. 결함 보고 활동은 이러한 테스트 실행 로그를 분석하여 결함들을 식별하고 이를 결함 보고서에 기록함으로써 검출된 결함의 해결을 시작할 수 있도록 한다. 그림 14.1은 테스트 실행 활동과 결함 보고 활동이 수행되는 세부 작업을 보여 준다.

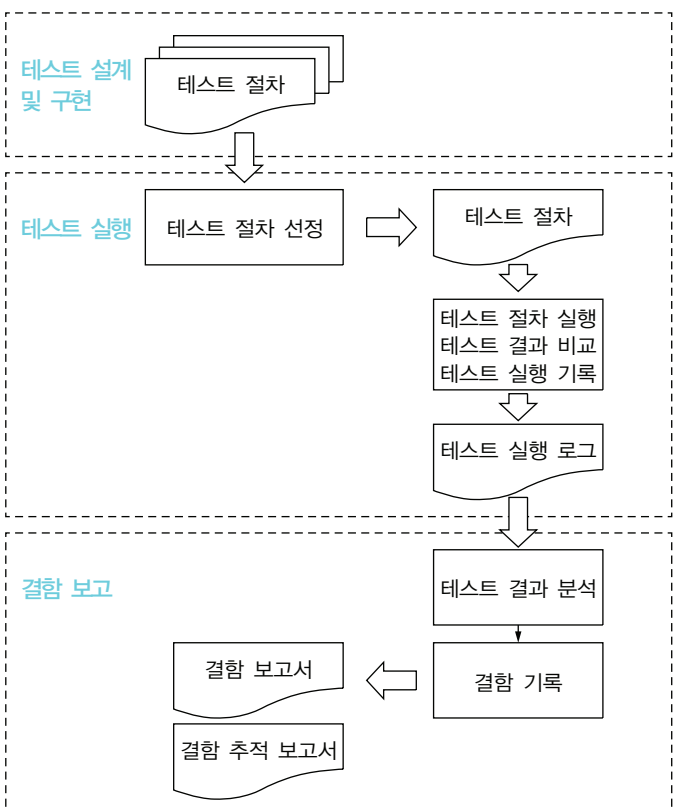

그림 14.1 테스트 실행 및 결함 보고

- **테스트 실행**: 주어진 테스트 절차 중에서 실행하고자 하는 테스트 절차를 선정한다. 그리고 선택된 테스트 절차를 실행하고 테스트 결과를 비교하며 그 결과를 테스트 실행 로그로 기록한다.

- **결함 보고**: 테스트 실행 로그를 바탕으로 테스트 결과를 분석하여 결함을 식별한다. 그리고 식별된 결함을 결함 보고서로 기록한다. 또한, 식별된 결함이 해결되고 종료될 때까지의 과정을 결함 추적 보고서에 기록한다.

표 14.1은 테스트 실행 활동과 결함 보고 활동을 수행하면서 작성하는 산출물을 보여 준다. 테스트 실행 활동에서는 설명, 테스트 작업과 이벤트 목록을 포함하는 테스트 실행 로그를 작성한다. 그리고 결함 보고 활동에서는 검출된 각 결함에 대하여 결함 보고서를 작성하고, 결함에 대한 처리(검토, 해결, 해결 검증 등) 과정은 결함 추적 보고서에 작성한다.

표 14.1 **테스트 실행 및 결함 보고 활동 산출물**

활동	산출물	설명
테스트 실행	테스트 실행 로그	테스트 실행 결과 테스트에 대한 전반적인 설명, 수행된 테스트 작업과 이벤트를 나열한다.
결함 보고	결함 보고서	검출된 각 결함에 대하여 결함 컨텍스트, 결함 설명, 심각도, 우선순위, 위험 분석, 결함 상태를 기술한다.
	결함 추적 보고서	보고된 각 결함이 종결될 때까지의 결함 검토 정보, 결함 해결 정보, 결함 해결 검증 정보를 기술한다.

14.2 테스트 실행

14.2.1 개요

테스트 설계 및 구현 활동에서 개발된 테스트 절차를 실행함으로써 테스트 실행 활동이 수행된다. 테스트 절차를 실행하였을 때 테스트 대상의 실제 수행 결과와 예상 결과를 비교하여 테스트 결과를 기록한다. 그림 14.2는 주어진 테스트 절차를 바탕으로 테스트 실행을 수행하는 세부 작업을 보여 준다.

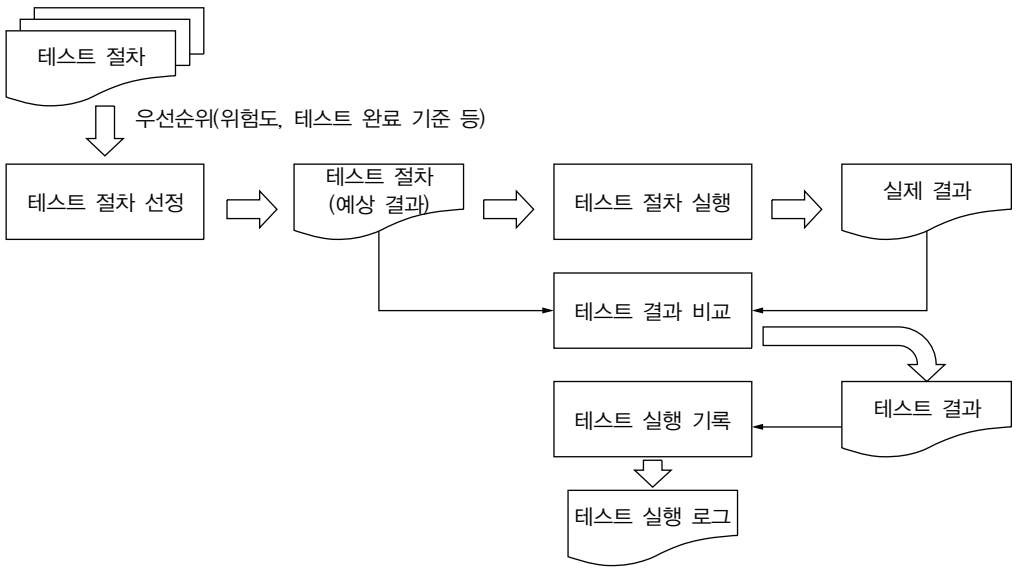

그림 14.2 **테스트 실행**

주어진 테스트 절차 중에서 우선순위를 고려하여 테스트 절차를 선택한다. 선택된 테스트 절차를 실행하여 관찰된 실제 결과와 예상 결과를 비교한다. 그리고 이러한 작업들을 테스트 실행 로그에 기록함으로써 결함 식별을 돕는다.

표 14.2는 테스트 실행 활동의 산출물을 보여 준다. 테스트 실행 활동에서는 수행된 테스트 실행 작업들을 테스트 실행 로그에 구체적으로 기록한다.

표 14.2 **테스트 실행 활동 산출물 요약**

산출물	주요 항목
테스트 실행 로그	• 설명 • 테스트 작업과 이벤트 목록

14.2.2 테스트 절차 선정

테스트 설계 활동에서 일반적으로 수많은 테스트 케이스 및 테스트 절차가 개발된다. 따라서 수많은 테스트 케이스들 중에서 어떤 것을 먼저 실행할지 결정이 필요하다. 테스트 케이스 실행 순서 결정은 우선순위를 이용한 방법과 테스트 완료 기준을 이용한 방법이 있다.

14.2.2.1 우선순위 전략

효율적인 테스트를 수행하기 위하여 피처 집합, 테스트 케이스, 테스트 절차마다 우선순위를 정의하고 있다. 따라서 테스트 절차를 선택할 때 이들 우선순위를 활용할 수 있다.

- **피처 집합 우선순위**: 각 피처 집합은 그 중요도에 따라서 우선순위가 부여되어 있다. 그러므로 테스트를 수행할 때는 우선순위가 높은 피처 집합에 테스트를 우선 수행하는 것이 바람직하다. 따라서 우선순위가 높은 피처 집합의 테스트 절차를 우선순위가 낮은 피처 집합의 테스트 절차보다 먼저 선택한다.
- **테스트 케이스 우선순위**: 각 테스트 케이스는 그 중요도에 따라서 우선순위가 부여되어 있다. 그러므로 테스트를 수행할 때는 우선순위가 높은 테스트 케이스를 먼저 사용한다.
- **테스트 절차 우선순위**: 각 테스트 절차는 그 중요도에 따라서 우선순위가 부여되어 있다. 그러므로 우선순위가 높은 테스트 절차를 먼저 선택하여 테스트한다.

14.2.2.2 테스트 완료 기준 전략

테스트 계획에서는 테스트 완료 여부를 판단하는 기준을 정의하였다. 테스트가 종료되면 이러한 테스트 완료 기준에 따라서 테스트 완료 여부를 평가하고 이를 테스트 종료 보고서에 기록한다. 따라서 테스트 완료 기준 달성에 가장 큰 기여를 할 수 있는 테스트 절차를 먼저 실행하는 것이 바람직하다. 표 14.3은 테스트 완료 기준에 따라서 테스트 절차를 선택하는 예를 보여준다.

표 14.3 **테스트 완료 기준별 테스트 절차 선택 예**

테스트 완료 기준		테스트 절차 선택
(1)	90% 이상의 모듈이 통과되어야 한다.	아직 통과되지 않은 모듈에 관한 테스트 절차 선택
(2)	TCS-10과 TCS-20 테스트 케이스는 통과되어야 한다.	TCS-10과 TCS-20을 포함하는 테스트 절차 선택
(3)	95%의 문장 커버리지가 충족되어야 한다.	문장 커버리지를 증가시킬 수 있는 테스트 절차 선택
(4)	심각한 결함이 존재하지 않아야 한다.	심각한 결함과 관련된 테스트 절차 선택

(1)과 같이 테스트 완료 기준이 테스트를 통과한 모듈의 비율이 일정 값 이상이 되어야 하는 경우에는 통과된 모듈의 비율을 높일 수 있는 테스트 절차를 먼저 선택해야 한다. 즉, 아직 통과되지 않은 모듈을 테스트 대상으로 하는 테스트 절차를 선택하는 것이 효율적이다. (2)와 같이 특정 테스트 케이스의 통과가 요구되는 경우에는 해당 테스트 케이스를 포함하는 테스트 절차를 우선 선택한다. (3)과 같이 문장 커버리지가 일정 비율 이상이 되어야 하는 경우에는 문장 커버리지를 높일 수 있는 테스트 절차를 선택한다. (4)와 같이 특정한 수준의 결함이 없어야 하는 경우에는 해당 수준의 결함을 검출할 수 있는 테스트 절차를 우선적으로 선택한다.

14.2.3 테스트 절차 실행

앞서 설명된 방법에 따라서 테스트 절차를 선택한 후에는 구축된 테스트 환경에서 선택된 테스트 절차를 실행한다. 테스트를 실행하는 주체는 테스트 레벨에 따라서 달라질 수 있다. 표 14.4는 각 레벨별 테스트를 실행하는 일반적인 주체를 보여 준다.

표 14.4 테스트 실행 주체

테스트 레벨	개발자	테스터	사용자
컴포넌트 테스트	⊙	⊙	
통합 테스트	⊙	⊙	
시스템 테스트	⊙	⊙	⊙
인수 테스트		⊙	⊙

컴포넌트 테스트는 개별적인 모듈이 테스트 대상이며 개발자가 테스트를 직접 실행시키는 것이 일반적이다. 이는 개발자가 개발자 환경에서 컴포넌트 테스트를 수행하면 테스트 환경의 구축비용을 줄일 수 있고, 테스트 결과를 정확하고 신속하게 이해할 수 있기 때문이다. 그러나 역할이 매우 중요한 모듈은 테스터가 참여하여 컴포넌트 테스트를 수행할 수도 있다.

통합 테스트는 개발자들이 직접 통합 테스트를 실행할 수 있고, 테스터가 진행할 수도 있다. 테스터가 통합 테스트를 수행하는 경우에는 더 전문적인 기술을 이용하여 객관적인 관점에서 체계적으로 테스트를 수행할 수 있다.

개발자들이 통합 테스트를 완료한 후에 개발자 환경에서 시스템 테스트를 수행할 수 있지

만, 테스터가 주도적으로 중요한 역할을 하는 테스트가 바로 시스템 테스트이다. 테스터는 일반적으로 요구사항 명세서를 바탕으로 명세 기반 테스트 기법을 사용하여 테스트 케이스를 생성하여 테스트를 수행함으로써 시스템이 요구사항 명세서를 충족하는지를 검증한다. 그리고 사용자가 시스템 테스트에 참여할 수도 있다. 이러한 경우, 좀 더 사용자 관점에서 테스트 케이스들이 시도될 수 있다는 장점이 있을 수 있다. 그리고 원칙적으로 인수 테스트는 사용자의 환경에서 사용자가 수행한다.

14.2.4 테스트 결과 비교

테스트 절차를 실행하여 테스트 절차를 구성하는 각 테스트 케이스에 대하여 예상 결과와 테스트 대상의 실행 결과를 비교한다. 예상 결과는 테스트 케이스에 명시되어 있거나 테스트 스크립트를 개발한 경우에는 스크립트에 내장되어 있거나 관련된 파일에 있을 수 있고, 탐색적 테스팅을 수행하는 경우에는 예상 결과가 문서화되지 않았을 수도 있다.

예상 결과와 실제 결과의 비교를 더 객관적이고 명확하게 수행하기 위해서는 예상 결과를 구체적으로 기술하는 것이 바람직하다. 예를 들어, 실숫값에 대해서는 비교가 필요한 유효 숫자의 자리를 정의하는 것이 바람직하다. 그리고 사용자 화면의 경우에는 화면을 구성하는 항목들의 배치, 각 항목의 색상 및 표시 메시지 내용 등도 구체적으로 정의되고 비교되어야 한다.

테스트 케이스에 따라서 비교될 결과는 화면뿐만 아니라 소리, 진동, 파일, 데이터베이스 테이블, 네트워크 등의 다양한 형태의 값이 될 수 있다. 그러므로 비교 값의 형태에 따라서 적합하게 예상 결과를 정의하는 것이 바람직하다. 또한, 테스트 결과의 비교를 지원하는 자동화 도구를 사용하는 것도 권장된다. 매우 빠른 속도로 출력이 발생하거나 많은 양이 출력되거나 매우 정교한 수준으로 엄격한 비교가 요구되는 경우에는 특히 도구의 도움이 필요하다.

14.2.5 테스트 실행 기록

테스터는 테스트 절차를 실행하는 과정에서 실제로 수행한 구체적인 작업과 목격된 이벤트들을 시간대별로 테스트 실행 로그 문서에 기록한다. 표 14.5는 테스트 실행 로그의 예를 보여 준다.

표 14.5 테스트 실행 로그 예

1. 테스트 실행 로그 식별자: TL-02
2. 설명
   테스트 대상: TI-1
   테스트 환경: Windows XP SP2, MM 2GB, HD 120GB

3. 테스트 작업과 이벤트

시간	테스트 실행 세부 작업	관련 결함
	테스터 A가 테스트를 시작함	
	시스템을 부팅함	
	TI-1을 실행함	
	TCS-01을 TP-01에 따라서 적용함	
	실행 결과가 TCS-01의 예상 결과와 다름	DR-01
	TI-1을 종료함	
	TI-1을 실행함	
	TCS-02를 TP-2에 따라서 적용함	
	실행 결과가 TCS-02의 예상 결과와 동일함	
	그러나 예상치 않게 파일-1이 삭제됨	DR-02
	…	…

❑ 설명

테스트 실행 로그 문서에서는 로그가 발생한 테스트 대상, 수행된 테스트 케이스, 적용된 테스트 절차 등을 간략하게 명시한다.

❑ 테스트 작업과 이벤트

시간대별로 테스터가 수행한 세부 작업을 기록한다. 일반적으로 테스트 대상 실행 준비 작업, 테스트 케이스를 입력하는 작업, 테스트 대상 실행 작업, 테스트 결과 관찰 작업 등이 기술된다. 기존의 테스트 케이스 및 테스트 절차를 그대로 적용하는 경우에는 해당 테스트 케이스 및 테스트 절차에 대한 식별자를 기술한다.

그리고 예상과 다른 결과가 관찰되거나 또는 전혀 예상치 않은 이벤트가 발생하면 해당 이벤트를 구체적으로 기술해야 한다. 만약, 이 이벤트를 바탕으로 결함을 식별하고 결함 보고서를 작성하였다면 작성된 결함 보고서의 식별자를 기록한다.

14.3 결함 보고

14.3.1 개요

테스트 실행 활동에서 적절한 조치가 필요한 이슈를 발견하면 결함으로 보고한다. 사실 결함을 포함하여 다양한 이슈가 식별될 수 있으므로 결함보다는 광범위한 의미인 인시던트(Incident)가 더 적합한 용어이겠지만, 테스트의 결과라는 측면에서 결함이라고 부르겠다. 참고로 ISO/IEC/IEEE 29119-2 표준에서는 테스트 인시던트 보고(Test Incident Reporting)라고 한다.

그림 14.3은 테스트 실행 활동의 결과물인 테스트 실행 로그를 바탕으로 결함 보고가 진행되는 세부 작업을 보여 준다. 테스트 실행 로그를 분석하여 결함을 식별하면 결함 보고서를 작성한다. 결함 해결이 효과적이고 효율적으로 수행되도록 결함의 구체화, 고립화, 일반화를 목표로 테스트 결과를 분석한다.

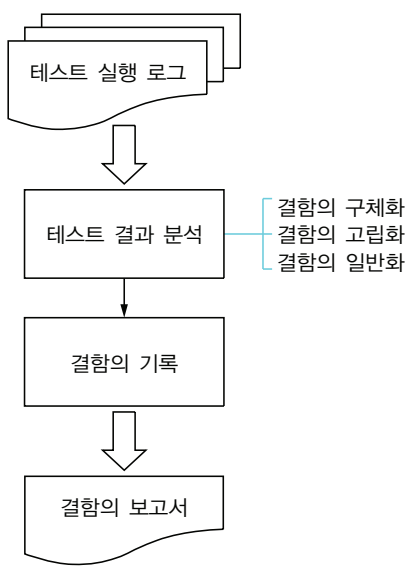

그림 14.3 결함 보고

테스트 실행 결과물에 대한 분석을 바탕으로 검출된 결함은 결함 보고서에 기록된다. 그리고 검출된 결함 수정, 수정 적절성 확인 등 일련의 작업을 결함 추적 보고서에 기록하고 개발자 등을 포함한 이해관계자와 의사소통한다. 표 14.6은 결함 보고 활동의 산출물인 결함 보고서와 결함 추적 보고서의 주요 항목을 보여 준다.

표 14.6 결함 보고 활동 산출물

산출물	주요 항목
결함 보고서	• 결함 컨텍스트 • 결함 설명 • 심각도 • 우선순위 • 위험 분석 • 결함 상태
결함 추적 보고서	• 결함 검토 정보 • 결함 해결 정보 • 결함 해결 검증 정보

14.3.2 테스트 결과 분석

테스터는 테스트 절차 실행을 통하여 발견된 결함을 추가적으로 분석하여 결함 발생 상황을 더욱 명확하게 파악해야 한다. 즉, 결함이 유발된 상황의 테스트 데이터, 기대하였던 값, 실제 관찰된 값과 테스트 절차 및 테스트 환경 등을 명확히 파악할 필요가 있다.

이는 결함 발생 상황이 구체적이고 명확히 정의되어야만 개발자가 결함을 통보받은 후 효율적으로 해결할 수 있기 때문이다. 이를 위해서 테스터는 테스트 실행 결과를 분석하여 결함을 구체화, 고립화, 일반화한 후에 해당 결함을 기록하는 것이 바람직하다.

14.3.2.1 결함의 구체화

개발자는 보고된 결함의 원인을 찾기 위하여 결함을 재연해야 한다. 따라서 발견된 결함을 재연(Reproduce) 가능할 정도로 결함 관련 테스트 데이터, 테스트 절차, 테스트 환경이 명확히 파악되어야 한다. 다시 말하면 결함을 검출할 때의 상황, 즉, 테스트 데이터, 테스트 절차, 그리고 테스트 환경을 구체화함으로써 발견된 결함을 재연할 수 있는 충분하고 명확한 상황을 파악해야 한다.

테스트 실행을 통하여 예상치 않은 결함을 발견한 경우에는 여러 회 동일한 테스트 실행을 시도하여 결함 발생과 관련된 상황을 구체적으로 파악한다. 그리고 지속적으로 일관되게 발생하지 않고 간헐적으로 발생하는 결함은 시도 횟수 대비 결함 발생 횟수의 정보도 함께 결정될 수 있도록 한다.

14.3.2.2 결함의 고립화

결함은 사용된 테스트 데이터, 적용된 테스트 절차, 구축된 테스트 환경 등에 따라서 발생할 수 있다. 결함의 발생에 직접적인 영향을 미치는 구체적인 상황이 효과적인 디버깅에 도움을 줄 수 있다. 예를 들어, 결함의 발생 원인이 사용된 입력값 때문인지, 적용된 테스트 절차인지, 또는 테스트 환경 때문인지를 자세하게 분석할 필요가 있다. 그림 14.4는 사용된 테스트 데이터, 테스트 절차, 테스트 환경 중에서 일부 요소에서만 결함이 발생하는 상황임을 보여 준다.

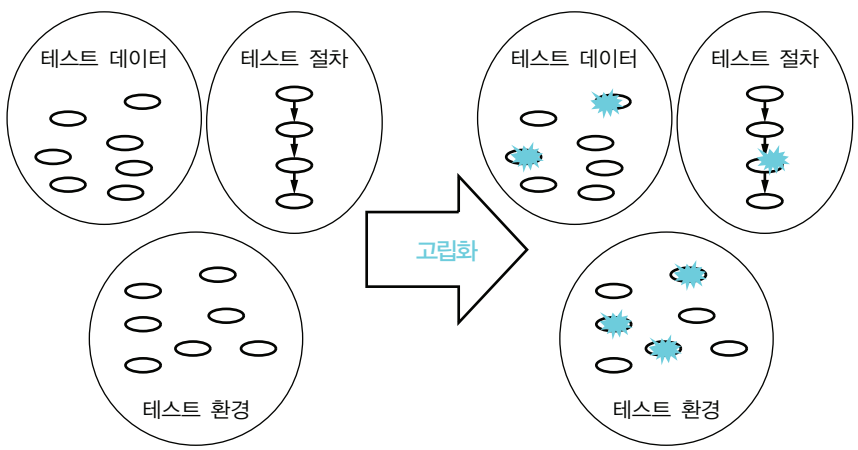

그림 14.4 결함의 고립화

따라서 결함이 발견되면 사용된 테스트 데이터, 테스트 절차, 테스트 환경을 구성하는 요소에 대해 어떤 요소가 결함 발생에 영향을 미치는지를 구체적이고 자세하게 분석한다. 결함 발생의 구체적인 상황을 파악하기 위해서는 결함의 발생에 영향을 미치는 것으로 추정되는 중요 요소를 바꿔가면서 테스트를 수행한다.

예를 들어, 결함의 구체적인 원인이 테스트 환경으로 추측된다면 테스트 데이터와 테스트 절차는 동일하게 사용하면서 다른 테스트 환경에서 테스트를 수행해 봄으로써 결함 발생의 근본적 요인이 테스트 환경에 있다고 생각할 수 있다.

14.3.2.3 결함의 일반화

결함의 발생에 영향을 주는 요소를 최대한 일반적으로 기술하는 것이 바람직하다. 예를 들어, 변수 X의 값이 −1, −2일 때 결함 발생을 발견하였다면 추가적인 분석/테스트를 통해서 X의 값이 −1, −2뿐만 아니라 음의 정수일 때 동일한 결함이 발생한다는 것을 파악할 수 있다.

또 다른 예로 윈도우 8 운영체제가 결함 발생의 요소라고 파악이 되었을 때 추가 테스트를 실행하여 윈도우 8뿐만 아니라 윈도우 7과 윈도우 10 등에서도 동일한 결함이 발생함을 발견하였다면 윈도우 8 운영체제 대신에 위의 모든 운영체제에서 결함이 발생한다고 판단하는 것이 타당할 것이다. 그림 14.5는 동일한 결함을 발생시킬 수 있는 여러 가지 상황을 조합함으로써 결함 발생 상황을 일반화하는 모습을 보여 준다.

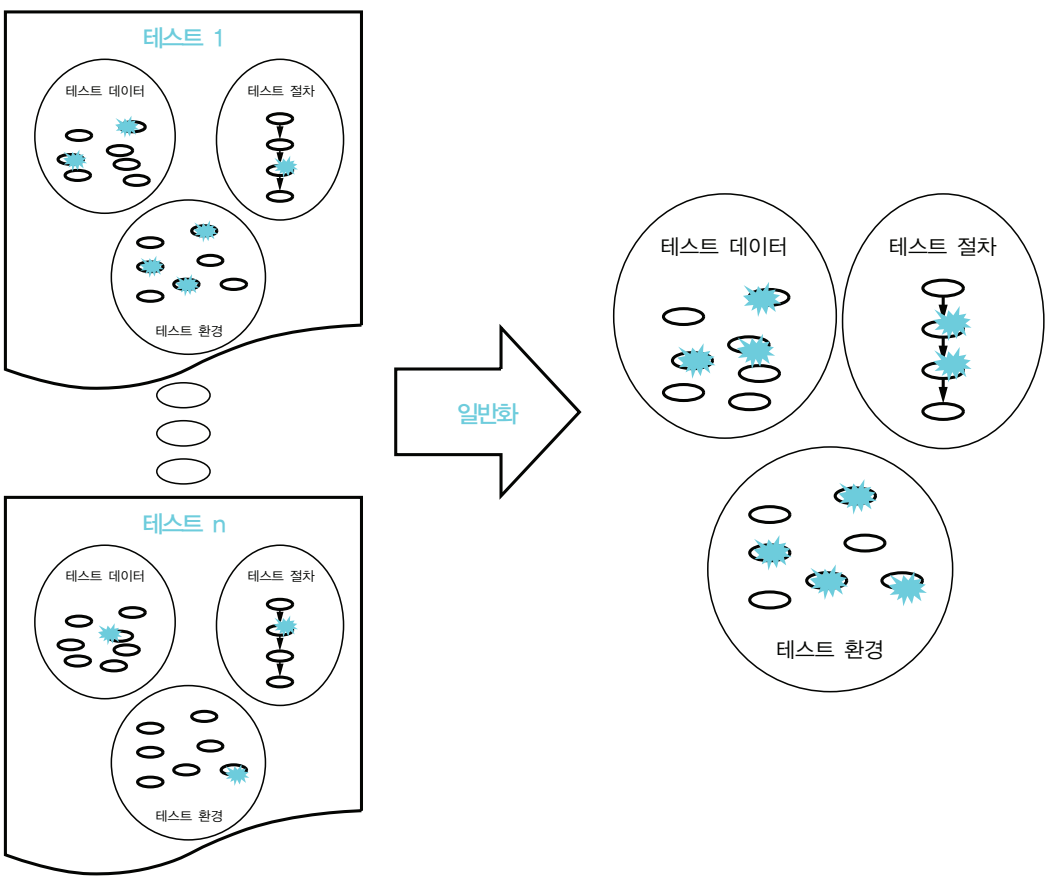

그림 14.5 결함의 일반화

14.3.3 결함 기록

테스트 결과 분석을 바탕으로 식별된 결함은 결함 보고서에 기록된다. 결함 보고서는 버그 보고서(Bug Report), 테스트 사건 보고서(Test Incident Report), 문제 보고서(Anomaly Report) 등으로도 불린다. 결함 보고서에 각 결함에 대하여 기록해야 할 중요한 항목은 다음과 같다.

- **결함 컨텍스트**: 어떤 상황에서 해당 결함이 식별되었는지 기술한다. 테스트 및 개발자가 결함을 정확히 이해하고 재연하는 것이 가능하고, 결함의 제거를 도울 수 있는 충분한 정보가 명확하게 기술되어야 한다.
  다음 사항을 포함하여 결함 발생 상황을 구체적으로 기술한다. 테스트 환경, 테스트 절차, 그리고 테스트 케이스는 해당 명세가 있다면 이에 대한 참조를 포함할 수 있다.

  – 개별 테스트: 결함을 검출한 개별 테스트를 명시한다. 즉, 컴포넌트 테스트 수행 중에 발견한 결함인지, 통합 테스트 중에 발견한 결함인지 등을 기술한다.
  – 테스트 대상: 결함이 검출된 테스트 대상을 고유하게 지칭한다.
  – 테스트 환경: 결함을 발생시킨 테스트 환경을 구체적으로 기록한다.
  – 테스트 절차 및 테스트 케이스: 결함을 발생시킨 테스트 실행 절차 및 테스트 케이스를 기술한다. 그리고 결함을 발생시킨 구체적인 테스트 데이터도 명시한다.

- **결함 설명**: 목격된 결함이 재연되고 해결될 수 있도록 상세하게 기술한다.
  – 실제 결과: 실제로 관찰된 결괏값을 기록한다. 결과가 화면으로 나오는 경우 화면을 캡처하거나, 파일로 출력 가능한 경우 파일로 결과를 기록할 수 있다. 결함을 고립화하고, 일반화하여 결함의 해결을 도울 수 있는 정보도 기술한다.
  – 이상 상황: 실제 결과와 예상 결과의 차이점에 대한 분석 내용 및 예상치 않게 발견된 오동작 상황을 기록한다.

- **심각도**: 발견자의 관점에서 볼 때 기술적인 측면과 비즈니스적인 측면을 모두 고려하여 검출된 결함이 미칠 수 있는 영향의 범위와 크기를 바탕으로 심각도를 기술한다. 결함 해결에 소요되는 예상 시간도 기술할 수 있다. 또한, 임시로 적용할 수 있는 우회 방법(Workaround)도 있다면 기술한다.

- **우선순위**: 검출된 결함 해결의 긴급성(Urgency)을 기술한다. 대부분의 조직은 3단계에서 5단계 정도로 우선순위를 부여한다. 예를 들어, 즉시 해결, 다음 릴리스에서 해결 등이 될 수 있다.

- **위험 분석**: 검출된 결함과 관련된 새로운 위험에 대한 분석 결과를 기술한다. 또는 기존 위험의 갱신 상태, 즉, 발생 가능성, 영향도에 따른 위험도의 변경이 있다면 이점도 기술한다.

- **결함 상태**: 검출된 결함에 대한 조치 상태를 기록한다. 이는 현재 상태로 Open, Assigned, Resolved, … 등이 될 수 있다. 결함 상태에 대해서는 바로 이어서 설명되는 결함 추적을 참고하기 바란다.

14.3.4 결함 추적

테스터가 테스트 절차를 실행하여 발견한 결함은 결함 보고서로 기록된다. 발견된 결함은 적절한 개발자를 선정하여 수정을 요청해야 한다. 그림 14.6은 소프트웨어 개발 과정에서 개발자와 테스터 간의 협력 관계를 보여 준다.

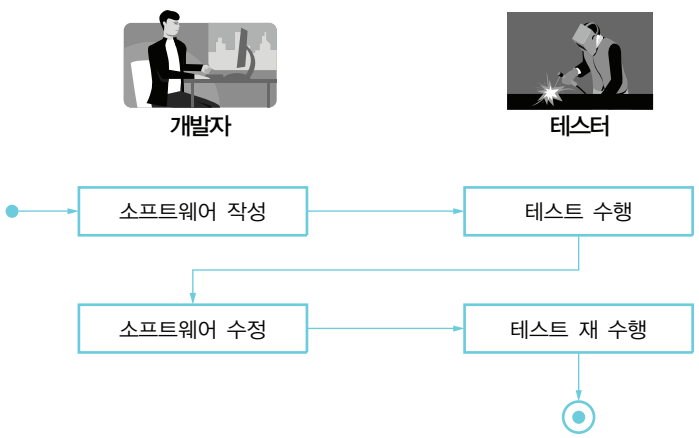

그림 14.6 개발자와 테스터의 협업

우선 개발자는 소프트웨어를 작성한 후 테스터에게 전달한다. 테스터는 계획된 테스트 전략에 따라서 테스트 케이스 및 테스트 절차를 생성하고 테스트를 수행한다. 테스터는 테스트 절차를 실행하는 과정에서 소프트웨어에 존재하는 결함을 검출할 수 있으며, 검출된 결함은 다시 개발자에게 통보되어 결함 수정을 요청한다. 개발자는 요청된 결함을 수정하고 다시 테스터에게 전달한다. 테스터는 요청된 결함이 정확하게 해결되었는지, 그리고 이 수정으로 인해 새로운 결함이 발생하지는 않았는지 점검하기 위하여 테스트를 실행한다.

14.3.4.1 결함 생명 주기

위 과정을 개발자와 테스터의 관점이 아니라 발견된 결함의 관점에서 보면 각 결함은 발견(Open)된 후에 종결(Closed)될 때까지 여러 가지 상황에 놓인다. 그림 14.7은 결함 생명 주기로 각 결함의 발견과 종결 사이의 상황 변화를 보여 준다. 결함 생명 주기를 구성하는 대표적인 결함의 상태로는 “Open”, “Review”, “Assigned”, “Resolved”, “Verified”, “Closed”, “Reopen”, “Deferred” 등이 있다.

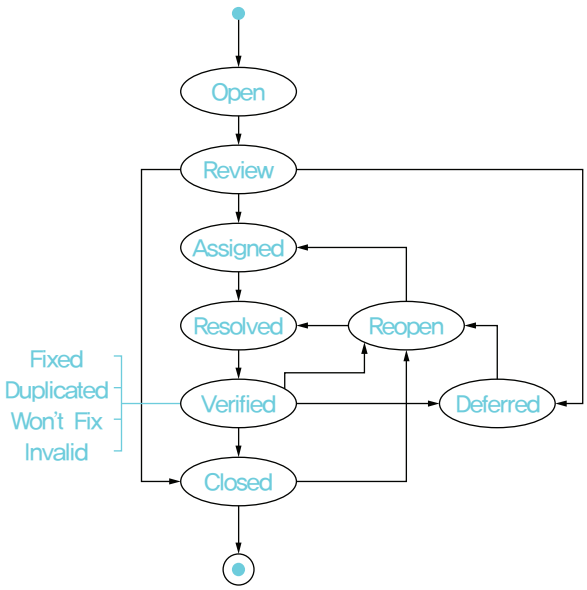

그림 14.7 **결함 생명 주기**

"Open" 상태는 테스터가 테스트 절차를 실행하여 발견한 결함을 분석 후 구체화, 고립화, 일반화한 결함으로서 보고된 상황을 뜻한다. 즉, 앞에서 소개한 결함 보고서에 기록되어 결함 추적의 대상이 된 상황을 뜻한다.

"Review" 상태는 "Open"된 결함의 처리 방안을 검토하는 상태이다. 각 결함은 위험성, 즉, 발생 가능성, 심각성, 긴급성을 바탕으로 이번에 수정되거나(Assigned 상태로 이동), 다음 릴리스에서 수정되거나(Deferred 상태로 이동) 또는 무시(Closed 상태로 이동)될 수 있다.

"Deferred" 상태는 "Open"된 결함을 곧바로 수정하지 않고 다음 릴리스에서 해결하기로 연기된 상태를 뜻한다. 그림에서 볼 수 있듯이 "Deferred"된 결함은 적절한 시점에 "Reopen"되어 결함 처리가 시작될 수 있다.

만약, 결함을 해결하기로 결정하였다면, 우선 수정 담당자가 결정되어야 한다. "Assigned" 상태는 결함을 수정할 개발자가 결정되고 그 개발자에게 결함 해결이 요구된 상태이다. "Resolved" 상태는 개발자가 자신에게 할당된 수정 해결을 처리한 상태이다. 요청된 결함을 처리하는 방법은 구체적으로 보면 다양한 경우가 있다.

개발자가 요청된 결함을 수정한 경우(Fixed), 요청된 결함이 기존의 다른 결함과 중복되는 경우(Duplicated)가 있을 수 있다. 그리고 개발자가 분석을 해 보니 지금 수정이 필요할 정도로 중요하거나 긴급한 것이 아니라서 수정을 하지 않은 경우(Won't Fix)도 있을 수 있다.

또한, 개발자가 분석을 해 보니, 프로그램의 문제가 아니라 결함 보고 자체가 문제가 있다고 판단할 수도 있다. 즉, 테스트 케이스 및 테스트 절차에 문제가 있는 경우(Invalid)가 있을 수 있다.

"Resolved" 상태는 결함 해결 요구에 대해 개발자의 처리 결과(Fixed, Duplicated, Won't fix, Invalid)만 제시된 상태이며, 개발자의 처리가 합당하거나 정확한지에 대한 검증은 이루어지지 않은 상태이다. 즉, 개발자가 "Fixed"라고 했지만 결함이 정확하게 수정된 것인지 아직 확인되기 전이다. 그리고 개발자가 "Invalid"라고 하였다면 실제 이 결함 보고 자체에 문제가 있는지 검증이 필요하다.

"Verified" 상태는 개발자의 결함 처리가 합당한지, 정확한지 검증이 된 상태이다. 검증 방법은 "Resolved"의 유형에 따라서 달라진다. 개발자가 "Fixed"라고 결함 수정을 보고하면 테스터는 결함이 정확하게 수정되었는지를 확인하기 위해서 재테스팅을 수행한다. "Duplicated"인 경우에는 개발자가 동일하다고 보고한 결함과 이번 결함이 동일한 것인지 그리고 동일한 이전 결함이 적절히 처리되었는지를 검증해야 한다. 만약 개발자가 "Won't Fix"라고 처리를 하였다면 테스트 관리자는 실제로 해당 결함이 중요하지 않거나 긴급한 처리가 필요하지 않은지 확인해야 한다.

"Verified" 상태 다음에 결함이 어떤 상태로 되는지는 검증 결과에 따라서 달라진다. 만약, "Fixed"에 대해서 정확한 수정이 이루어졌다고 판단이 되면 결함은 "Closed"로 이동하게 된다. 그러나 결함이 정확하게 수정되지 않았다면 "Reopen" 상태로 이동하여 다시 수정을 요구할 수 있다. 표 14.7은 개발자가 처리한 네 가지 유형의 결함 처리에 대해서 테스터가 검증하고 난 이후의 상태를 요약하여 보여 준다.

표 14.7 결함 처리 유형별 Verified 이후의 상태

Verified 이후의 상태	정확	부정확
Fixed	Closed	Reopen
Duplicated	Closed	Reopen
Won't Fix	Deferred	Reopen
Invalid	Closed	Reopen

14.3.4.2 결함 처리 시나리오

결함이 Open된 후에 Closed되는 시나리오는 다양하다. 대표적인 세 가지 시나리오를 설명하면 다음과 같다.

14.3.4.2.1 경미한 결함의 무시 시나리오

발견된 결함을 검토(Review)하였을 때 매우 경미한 결함이고 시스템에 미치는 영향이 무시될 수 있다고 판단되는 경우에는 "Open" – "Review" – "Closed" 상태로 이동될 수 있다. 그림 14.8은 경미한 결함이 어떤 상태로 이동하는지를 보여 준다.

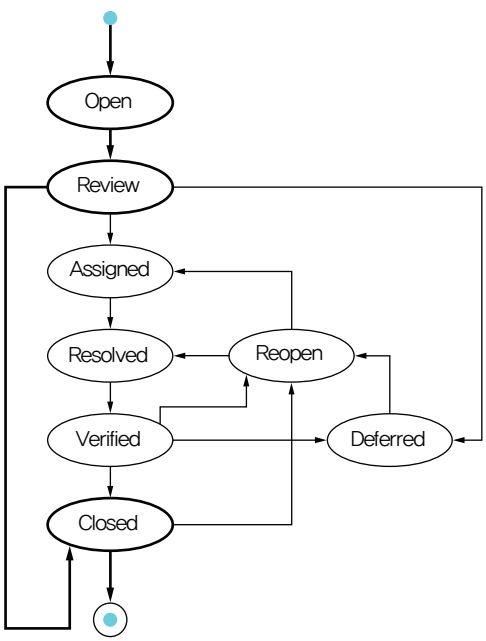

그림 14.8 경미한 결함은 무시되는 시나리오

14.3.4.2.2 차후에 결함을 처리하는 경우

발견(Open)된 결함을 검토(Review)하였을 때 다음 릴리스에서 적절한 처리를 하는 것이 바람직하다고 판단되면 "Open" – "Review" – "Deferred" 상태로 이동될 수 있다.

다음 릴리스에 대한 테스트를 수행할 때 "Deferred"에서 "Reopen"이 된 후에 담당 개발자가 결정된다("Assigned" 상태). 그러면 담당 개발자가 적절하게 처리("Resolved" 상태)를 한다. 표 11.7에서 볼 수 있듯이 개발자의 처리에 대한 검증("Verified" 상태) 결과에 따라서

결함은 수정(Fixed)되어 종료되거나("Closed" 상태), 다시 다음에 처리하기로 "Deferred" 될 수 있다. 그림 14.9는 차후에 처리하기로 미루어진 결함이 처리되는 시나리오를 보여 준다.

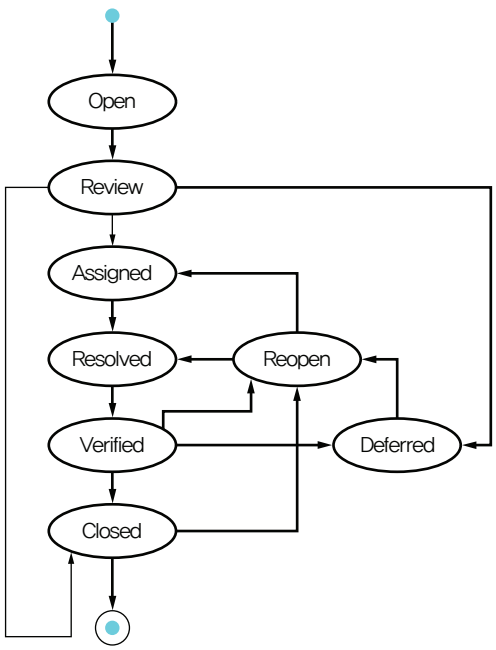

그림 14.9 차후에 결함을 처리하는 시나리오

14.3.4.2.3 이번 릴리스에서 결함을 처리하는 경우

발견된 결함을 검토(Review)하였을 때 이번 릴리스에서 처리하는 것이 바람직하다고 판단되면 해당 결함을 처리할 담당 개발자를 결정하고 수정을 요청한다. 즉, 결함은 "Open" – "Review" – "Assigned" 상태로 이동될 수 있다. 개발자는 수정 요청을 처리하며("Resolved"), 처리 유형과 처리 결과에 따라서 이후의 결함 상태가 결정된다(표 14.7 참고).

그림 14.10은 이번 릴리스에서 결함을 처리하는 시나리오를 보여 준다. 그림 14.9와 비교하면 차후 릴리스에서 결함을 처리하는 시나리오와 거의 동일하다. 다만 그림 14.9에서는 차후에 결함을 처리하므로 "Review" 상태에서 "Deferred"로 상태가 이동하는 반면에 그림 14.10은 이번 릴리스에서 처리를 하는 경우이므로 "Review" 상태에서 바로 "Assigned" 상태로 이동된다.

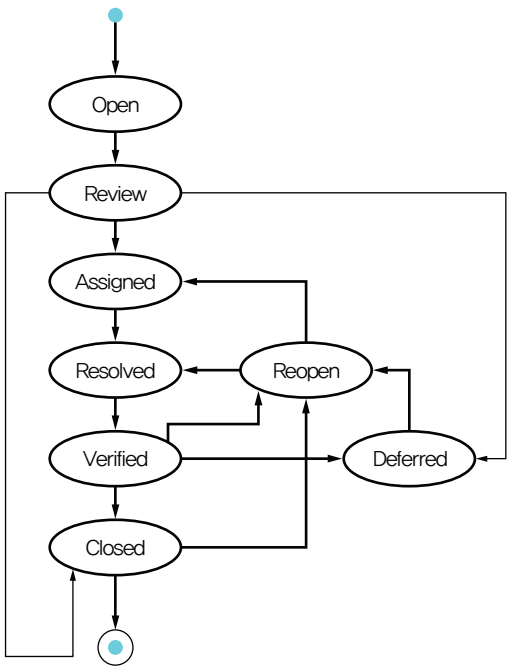

그림 14.10 이번 릴리스에서 결함을 처리하는 시나리오

14.3.4.3 결함 추적 보고서

지금까지 설명한 것처럼 테스터가 검출한 결함은 개발자를 할당하여 결함을 해결하거나, 다음 버전으로 미루거나 심지어 해당 결함을 무시할 수도 있다. 이렇게 결함의 등록을 시작으로 최종적으로 결함을 종결할 때까지 개발자 및 테스터가 수행한 작업을 명확하게 기록해야 한다.

표 14.8은 결함 보고서에 보고된 각 결함에 대한 검토, 해결, 그리고 해결에 대한 검증 작업을 기록하는 결함 추적 보고서 양식의 예를 보여 준다. 검토 정보는 보고된 결함에 대한 처리 유형으로서 1) 담당자 할당, 2) 처리 연기, 3) 결함 무시를 기록한다. 결함 해결 정보는 담당자가 자신에게 할당된 결함에 대한 해결 결과를 기록한다. 그리고 결함 해결 검증 정보는 담당자의 해결 결과의 적절성에 대한 검증을 수행한 결과를 기록한다.

표 14.8 결함 추적 보고서 양식 예

결함 검토 정보			
결함 식별자			
검토자		검토 일자	
처리 유형	☑ 담당자 할당 ☐ 처리 연기 ☐ 결함 무시		
검토 설명			
결함 해결 정보			
해결자		해결 일자	
해결 유형	☑ Fixed ☐ Duplicated ☐ Won't Fix ☐ Invalid		
해결 설명			
결함 해결 검증 정보			
검증자		검증 일자	
검증 결과			
검증 설명			

결함 추적 보고서는 결함을 발견한 테스터, 결함 해결을 할당받은 개발자, 결함 수정을 검증하는 테스터 등 일반적으로 다수 이해관계자에게 공유될 필요가 있다. 따라서 결함의 등록과 이에 대한 처리 과정을 다수 이해관계자가 쉽게 공유할 수 있도록 문서 대신에 결함 추적 지원 도구를 이용하는 것이 효과적이다.

테스트 자동화 도구의 일종으로 결함 추적 지원 도구는 결함의 기록과 결함 처리 및 검증 기록 그리고 결함의 처리 상황을 다양한 기준으로 분석하여 보고서를 작성하는 기능을 제공한다. 결함 추적 지원 도구를 이용하면 현재 진행 중인 테스트의 상황을 종합적으로 파악할 수 있다.

14.4 산출물 요약

14.4.1 테스트 실행 산출물

14.4.1.1 테스트 실행 로그

표 14.9는 테스트 실행 로그 산출물의 구성을 보여 준다. 테스트 실행을 수행한 테스터와 테스트 실행의 컨텍스트 측면에서 테스트 대상, 테스트 환경 등을 설명한다. 그리고 시간 대별로 실제 수행된 테스트 작업과 그 결과를 기술한다. 특히, 결함이 식별된 경우는 해당 결함에 대한 식별자를 기술한다.

표 14.9 테스트 실행 로그 구성

테스트 로그 식별자		
테스터		
설명		
테스트 로그		
시간	세부 작업	관련 결함

14.4.2 결함 보고 산출물

14.4.2.1 결함 보고서

표 14.10은 결함 보고서 산출물의 구성을 보여 준다. 결함 보고서는 결함 식별자, 결함을 발견한 테스터와 발견 일시를 기록한다. 그리고 결함을 발견할 때 수행된 구체적인 개별 테스트, 테스트 대상, 테스트 환경, 테스트 케이스 및 테스트 절차 등을 결함 컨텍스트에 기술한다. 실제 실행 결과 및 발견된 이상 결과를 결함 설명에 기술한다. 그리고 결함의 심각도, 우선순위, 관련 위험 분석 결과와 현재 결함의 상태를 기술한다.

표 14.10 결함 보고서 구성

결함 식별자	
테스터	
발견 일시	
결함 컨텍스트	
결함 설명	
심각도	
우선순위	
위험 분석	
결함 상태	

14.4.2.2 결함 추적 보고서

표 14.11은 결함 추적 보고서 산출물의 구성을 보여 준다. 결함 추적 보고서는 보고된 결함을 해결하고 종결시킬 때까지의 처리 과정을 포함한다. 참고로, 결함 추적 보고서는 ISO/IEC/IEEE 29119-3 표준에 포함되어 있지는 않다.

결함 추적 보고서는 추적 대상이 되는 결함의 식별자를 먼저 작성한다. 그리고 결함에 대한 검토 정보, 결함 해결 정보, 결함 해결 검증 정보를 기술한다. 결함 검토 정보는 결함 보고서를 바탕으로 Assigned, Deferred, Closed의 검토 결과를 기술하고 그러한 결과에 대한 기준과 Assigned인 경우에는 할당된 개발자를 검토 설명에 기술한다. 결함 해결 정보는 할당된 개발자가 어떻게 결함을 해결하였는지를 기술한다. 결함에 대한 판단은 Fixed, Duplicated, Won't Fix, Invalid 등이 될 수 있으며 상세한 내용을 해결 설명에 기술한다. 이러한 해결에 대한 검증 결과는 해결 검증 정보에 기술한다. 즉, Fixed가 제대로 되었는지를 확인하기 위하여 수행한 재테스팅 결과를 기술한다. 또는 Duplicated, Won't Fix, Invalid인 경우에도 이러한 판단이 타당하고 정확한지 검증한 결과를 기술한다.

표 14.11 결함 추적 보고서 구성

결함 검토 정보			
결함 식별자			
검토자		검토 일자	
처리 유형	☑ 담당자 할당 ☐ 처리 연기 ☐ 결함 무시		
검토 설명			
결함 해결 정보			
해결자		해결 일자	
해결 유형	☑ Fixed ☐ Duplicated ☐ Won't Fix ☐ Invalid		
해결 설명			
결함 해결 검증 정보			
검증자		검증 일자	
검증 결과			
검증 설명			