소프트웨어 검증

First Presentation

Presentation Topic

Eclipse, JUnit, Clover, JDepend

T3
200312489 유현덕 200711453 류진렬
200711477 황진수 200711428 박기성
201360220 황 민

Contents

Eclipse
소개

Eclipse 소개

» Eclipse

- 다양한 플랫폼에서 사용할 수 있는 범용 응용 소프트웨어 플랫폼
- JAVA 언어로 작성되었으며 다양한 배포판이 존재한다.
  (C/C++ 개발자용, 자바 개발자용, 웹 개발자용)
- RCP를 포함한 상위의 모든 기능을 제공하기 위해 '경량화된 소프트웨어 컴포넌트 프레임워크'인 플러그인을 사용한다.

TDD
소개

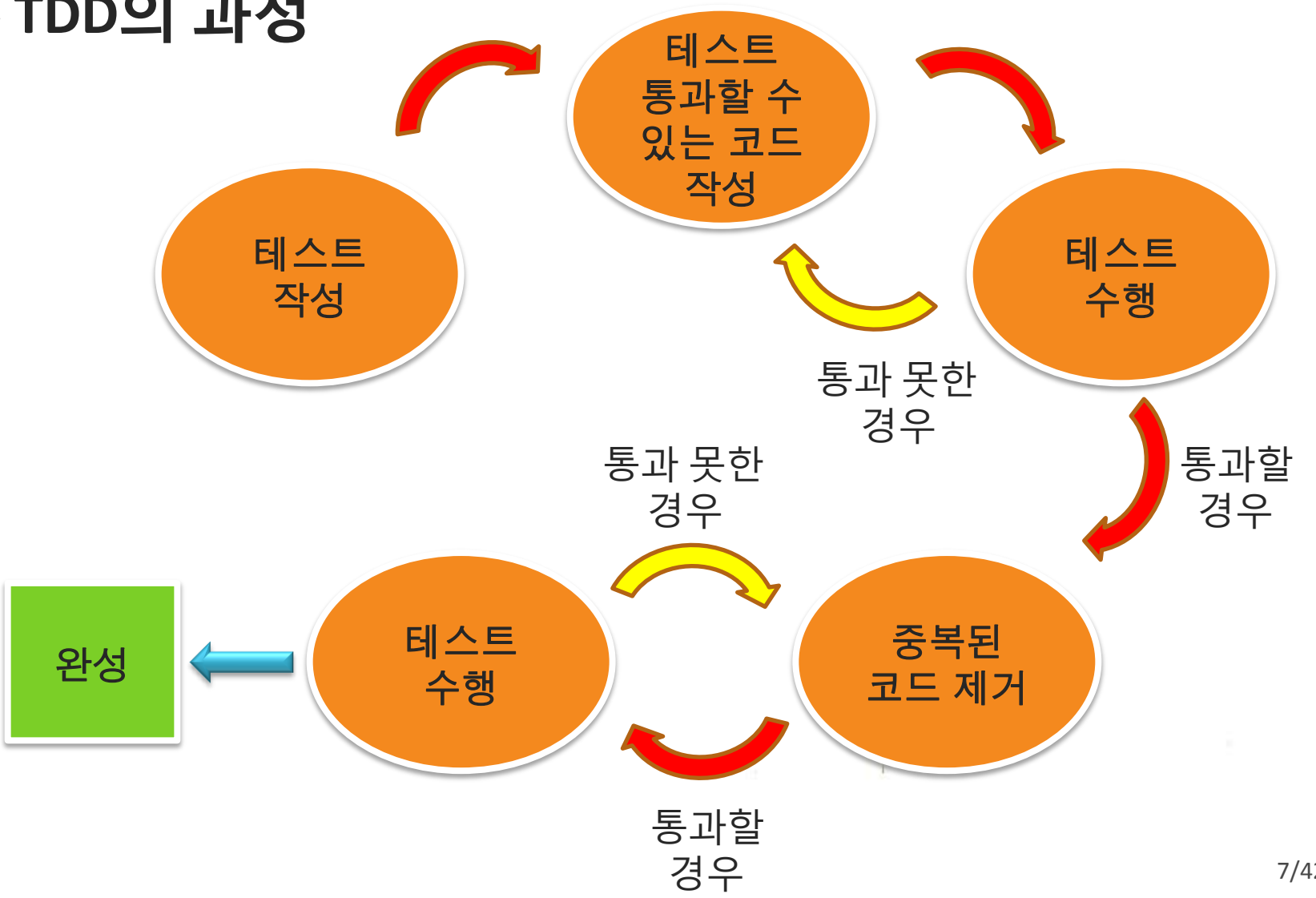
» Test Driven Development란?

- 작성해야 하는 프로그램에 대한 테스트를 먼저 작성하고 이 테스트를 통과할 수 있도록 실제 코드를 작성하는 개발 방법

- Ex)

```
int testAdd() {
        assert 5 + 4 == add(5, 4)
        assert -4 + 9 == add(-4, 9)
}
```

» TDD의 과정


테스트 통과할 수 있는 코드 작성
테스트 수행
테스트 작성
통과 못한 경우
통과할 경우
중복된 코드 제거
테스트 수행
통과 못한 경우
완성
통과할 경우

» TDD를 하는 이유

- 테스트 되지 않는 코드가 없어진다.
  -> 모든 코드가 테스트 되어 버그 발생 가능성을 줄임.

- 테스트 자체가 요구사항을 분명히 드러나게 한다.
- 자연스럽게 프로그램의 디자인이 SimpleDesign화 된다.
- 여러 변경 작업시 그 변경으로 인한 문제 발생을 쉽게 알아차린다.
  -> 개발 과정의 유연성을 높임
- 최종적으로 생산성 향상에 기여

Unit Test 소개

Unit Test(단위 테스트)란?

- 코드 내의 부분적인 단위 코드에서 문제 발생 가능성이 있는 부분을 테스트 하는 작업
- 특정 영역의 테스트를 위한 코드 조각(테스트 케이스)을 작성한다.
- 특정 상황에서 특정 메소드를 테스트한다.

Unit Test의 지향점

- 모든 메소드를 테스트 하는 것이 아니라 상식 수준의 확인을 통해 단위 코드가 의도한 대로 동작하는지 여부를 판단
- 코딩 전에 테스트케이스를 작성하여 구현 시 보조자료로 활용하는 것이 이상적

JUnit
소개

JUnit 소개

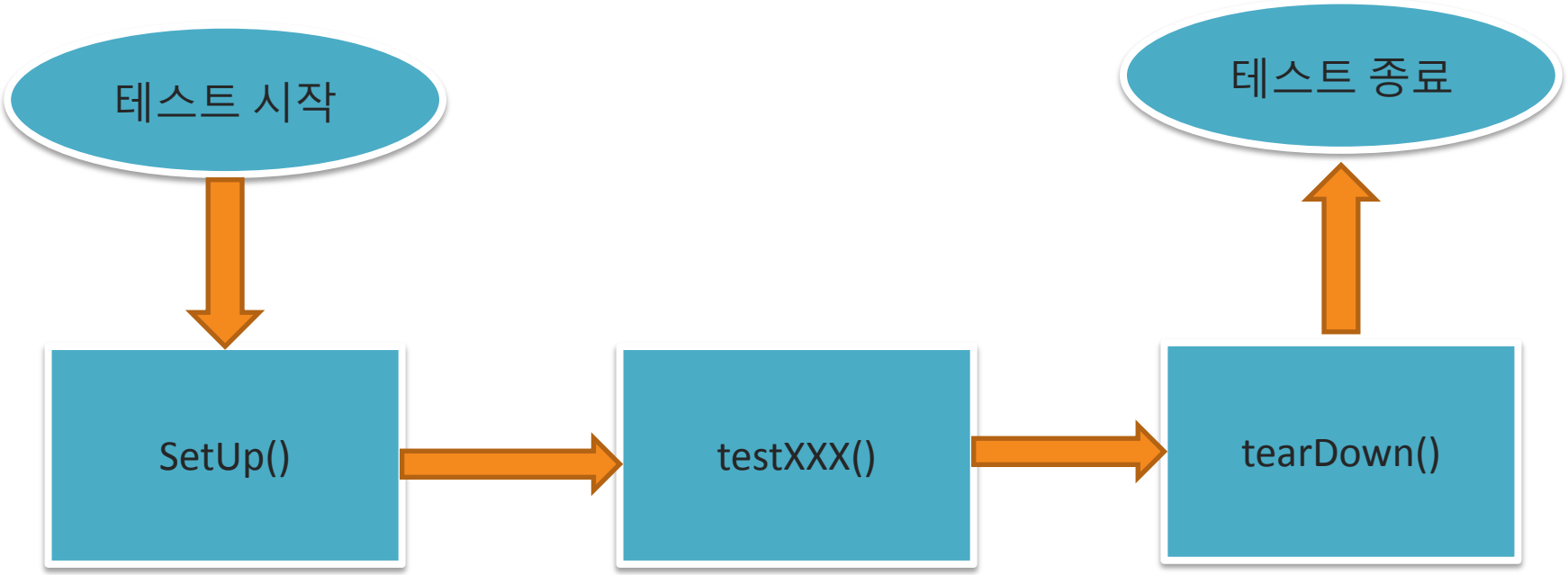
JUnit이란?

- TDD의 필수 조건 -> 테스트의 자동화
- 테스트 자동화에 사용하는 도구
- 가장 많이 사용되는 JAVA 단위 테스트 프레임워크

Junit이 필요한 이유

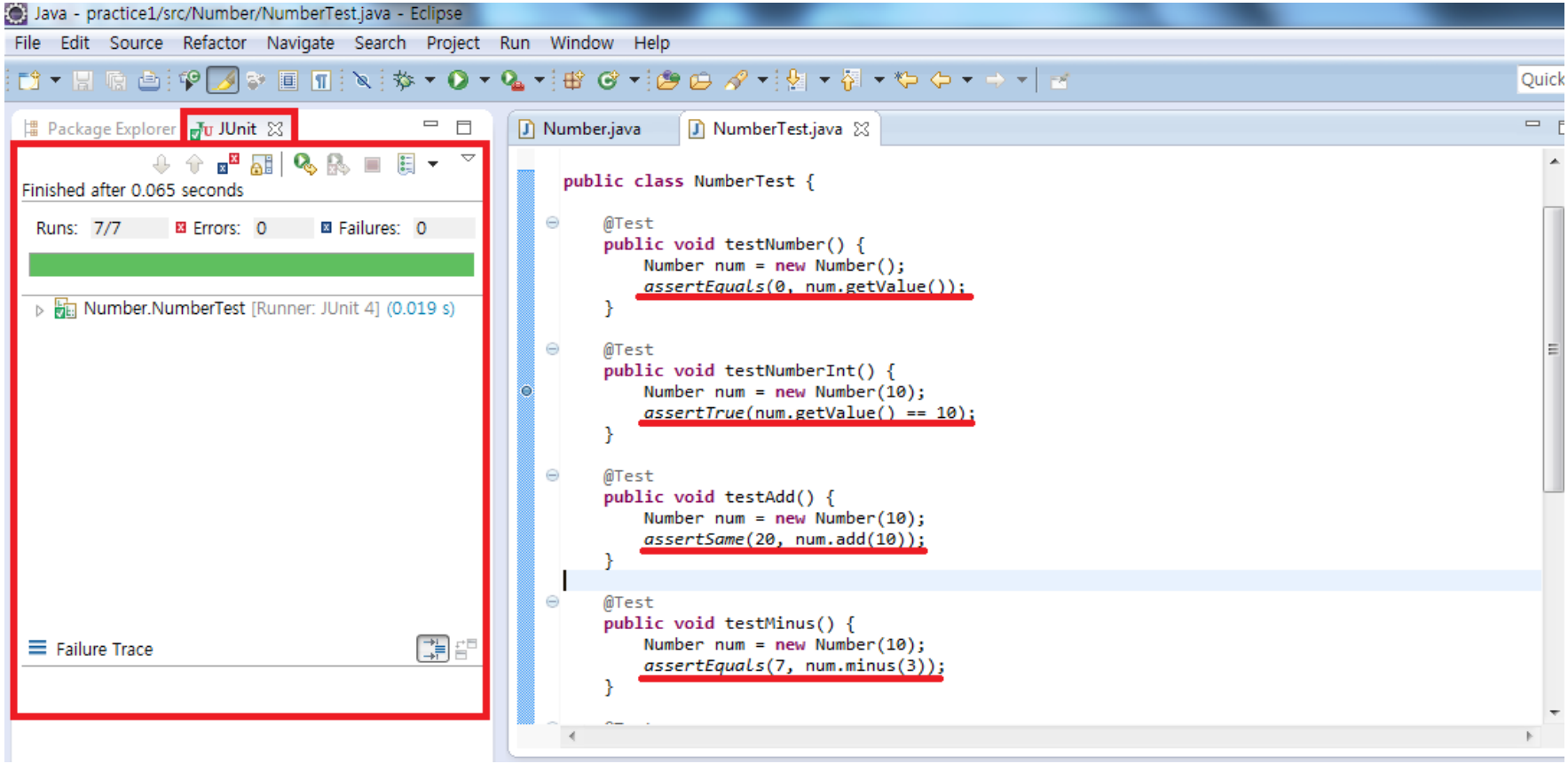
- 테스트 코드를 소스 코드와 분리 가능.
- 기존 테스트의 경우 한 번의 단일테스트가 실패 할 경우, 후속 테스트 불가능 -> 한번의 실행으로 다양한 테스트케이스를 종합적으로 확인 가능

Junit의 TestCase 수행 순서



JUnit 소개

» Junit의 TestCase 예제 (Junit 3)



Coverage
소개

Coverage 소개

» Test Coverage

- 개발자가 테스트를 충분히 했는지 확인할 수 있는 방법
- 테스트할 때 소프트웨어를 얼마나 실행했는지 분석한 값
- 정적 테스트 방법
- 명세 기반 테스트 커버리지 =
  테스트한 명세 개수 / 전체 명세 개수 * 100

» Code Coverage

- 명세 기반 테스트 커버리지가 의미를 갖지 못할 때 이를 보완할 수 있는 방법
- 동적 테스트 방법
- 신뢰성이 높지만 자체만으로 안정성을 보장하지 못함
- 코드 커버리지 =
  테스트된 코드 구성 요소 개수 / 전체 코드 구성 요소 개수 * 100

» Code Coverage 구분

- 구문 커버리지
- 결정 커버리지
- 조건(분기) 커버리지
- 조건 / 결정 커버리지
- 변형조건 / 결정 커버리지
- 다중 조건 커버리지

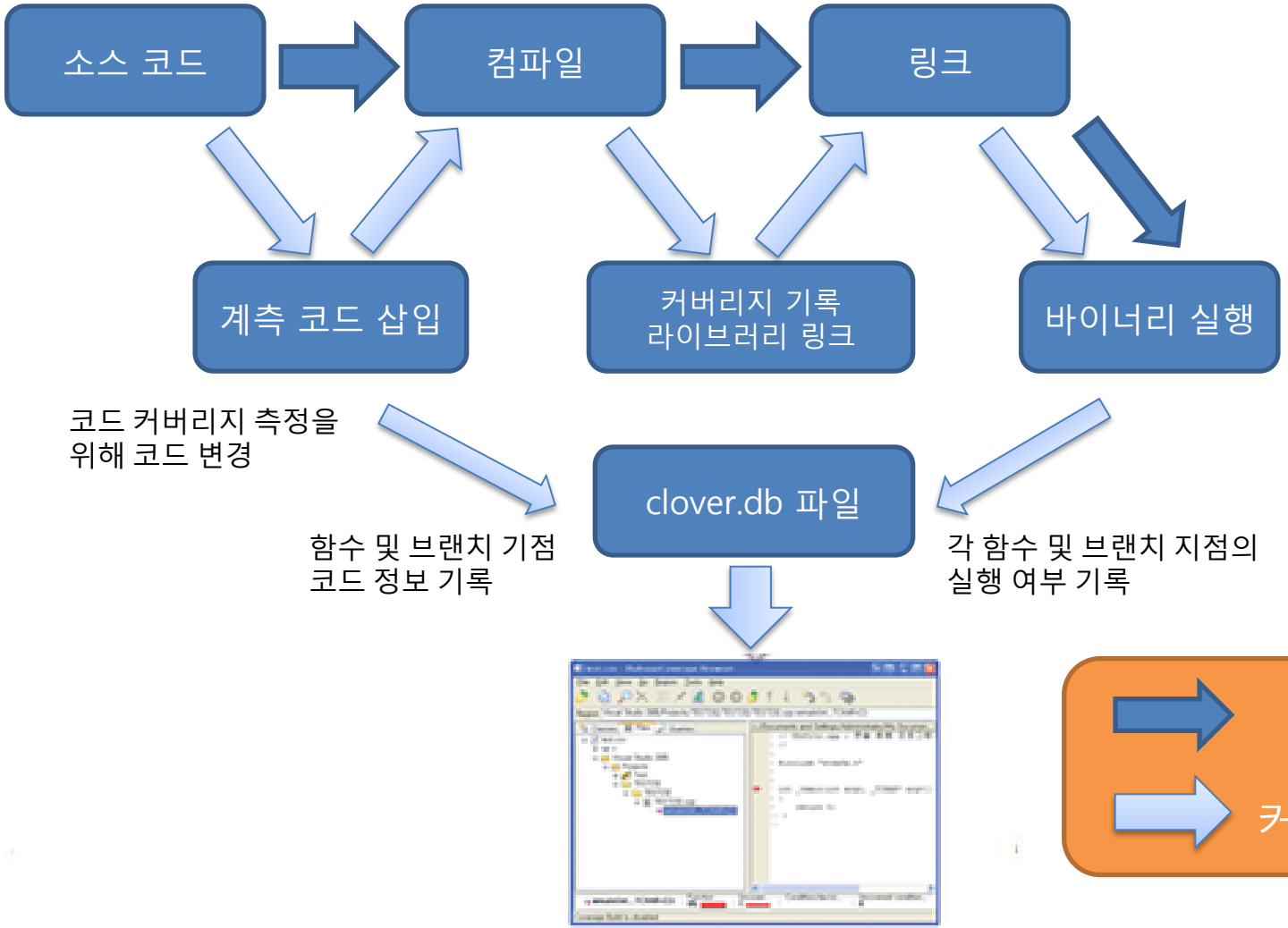
Clover 소개

» Clover 사용 목적

- 테스트에 의해 프로그램이 어떻게 작동하는지 확인
- 전체 테스트 부분 중 얼만큼 완료됐는지 판단
- 결과에 따라 테스트 기능 개량 가능
- 검사 항목, 기능 추가 여부 결정

Clover 소개

» **Clover 작동 원리**



clover.db 파일의 내용을 HTML이나
XML 등의 형식으로 출력

» Clover 사용하기

» 1. 적용한 프로젝트 : Network Pacman Game

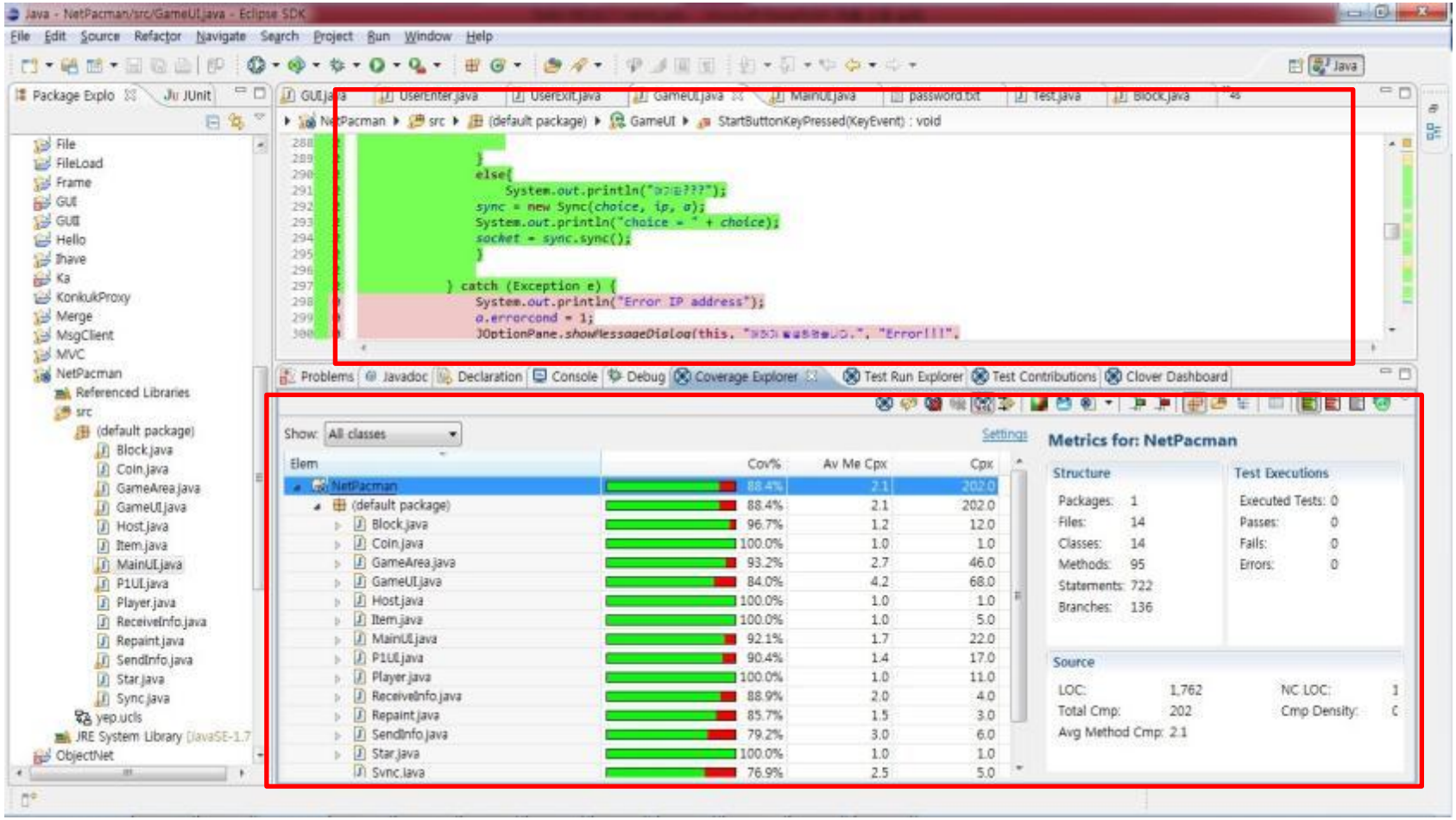
- Host Player와 Guest Player가 TCP/IP 연결을 통해

대전 팩맨 게임을 할 수 있는 프로그램.

Clover 소개

» Coverage Explorer View

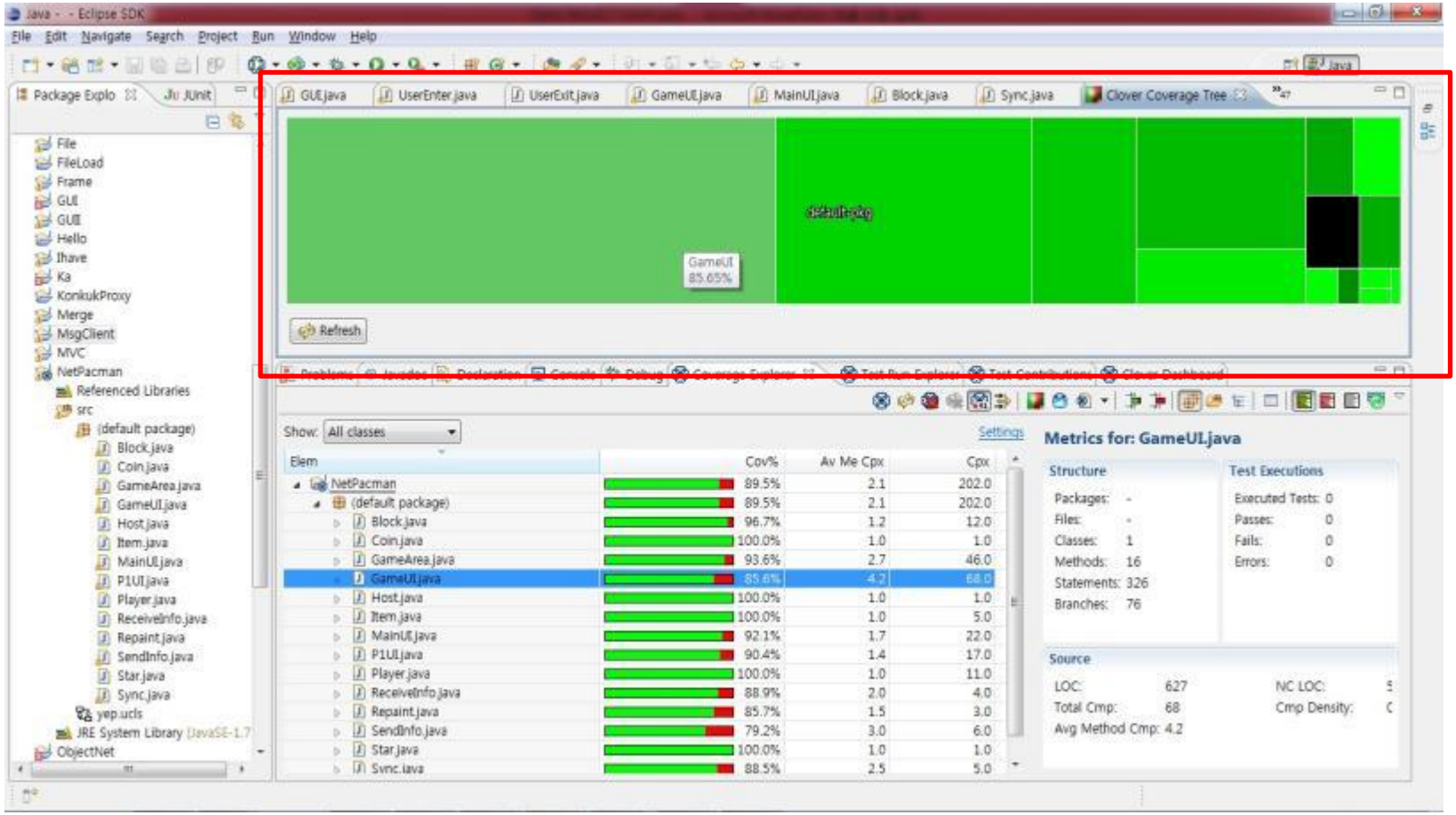
- **Element, Coverage(%), Av Me Cpx, Cpx.**



Clover 소개

» Tree map Report

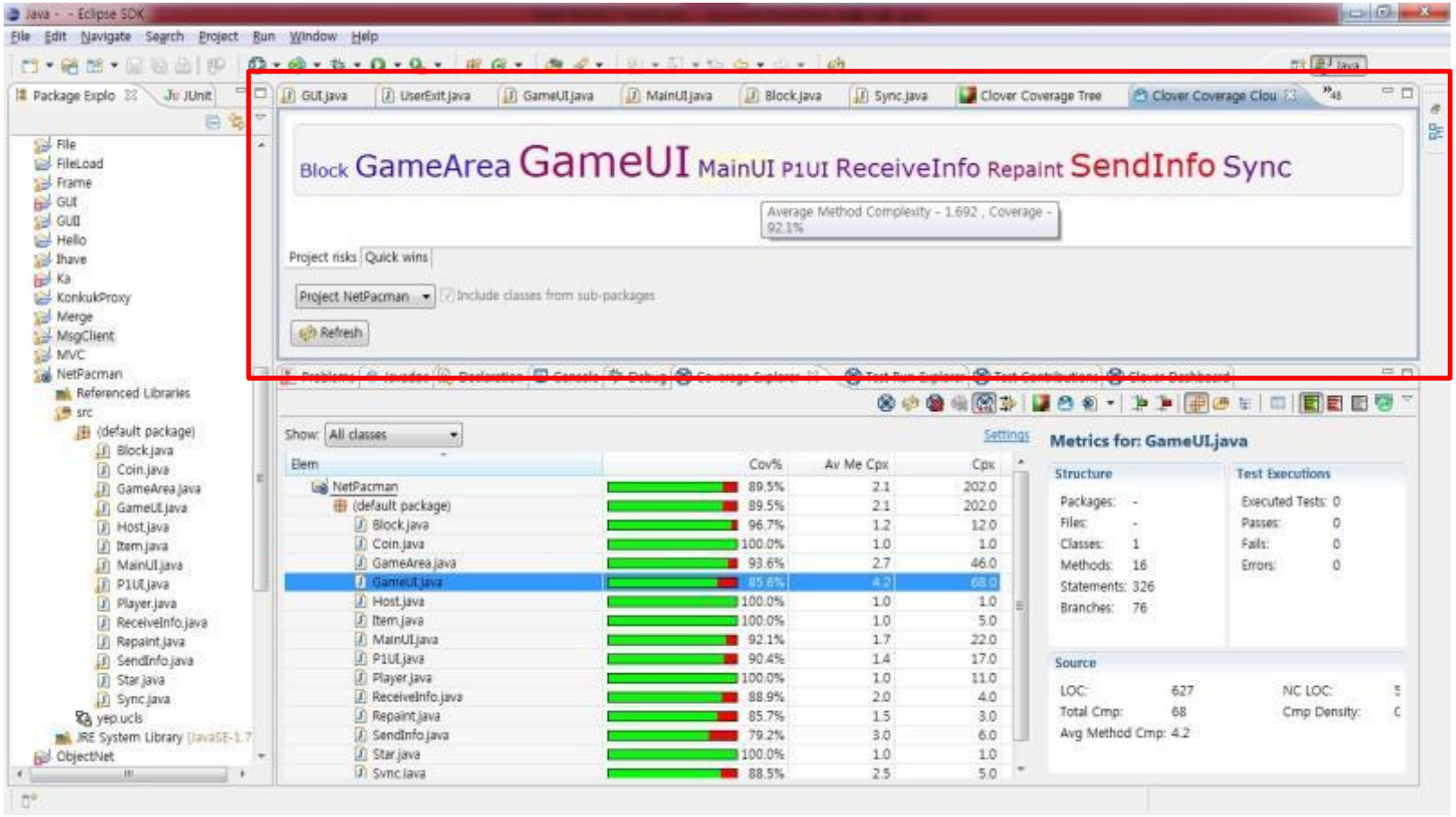
- Tree map 형태로 커버리지율을 표시



Clover 소개

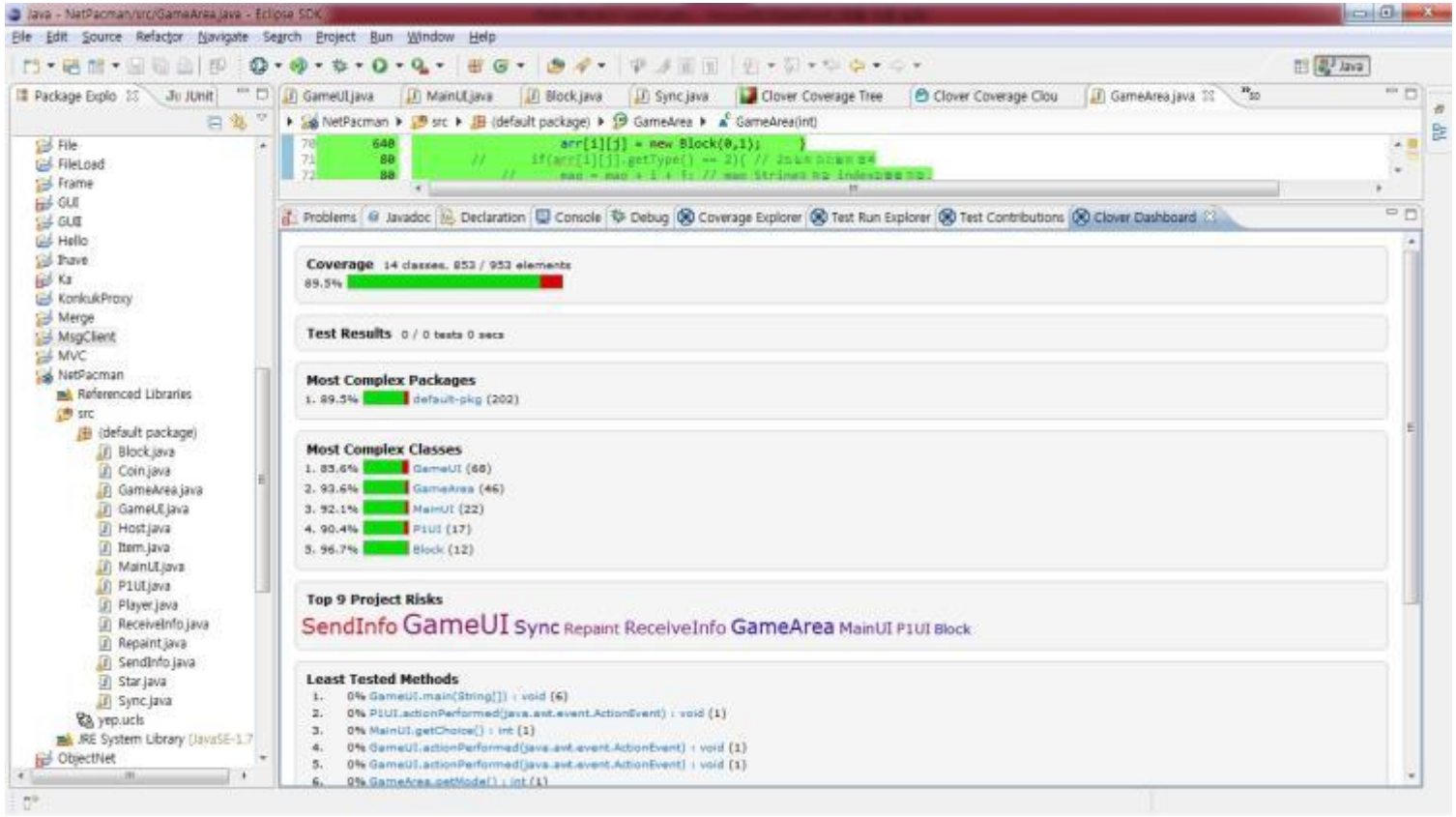
» Cloud Report

- **Cloud 형태로 Risk를 표시**



» Clover Dashboard

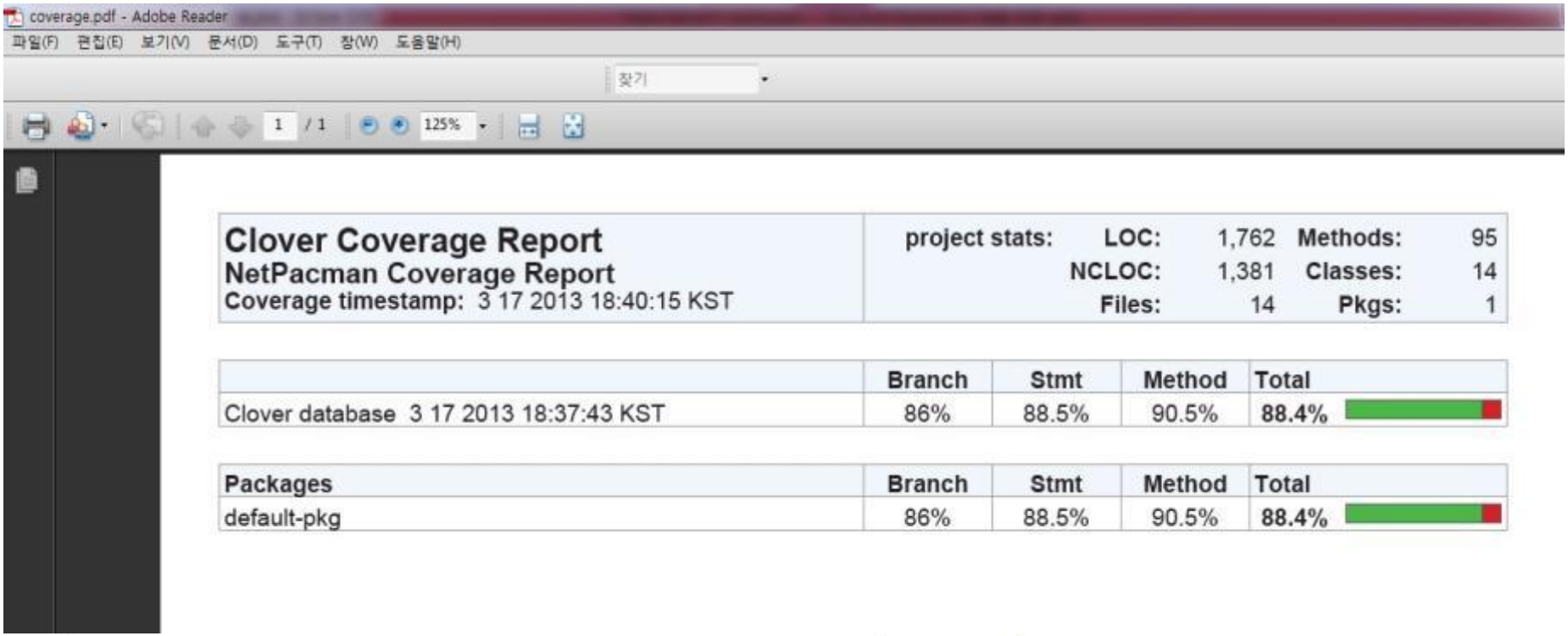
- Coverage, Test Result, Most Complex Package, Project Risks, Least Tested Methods



Clover 소개

» PDF, HTML, XML Report

- pdf, html, xml과 같은 형태로 report 뽑아내기



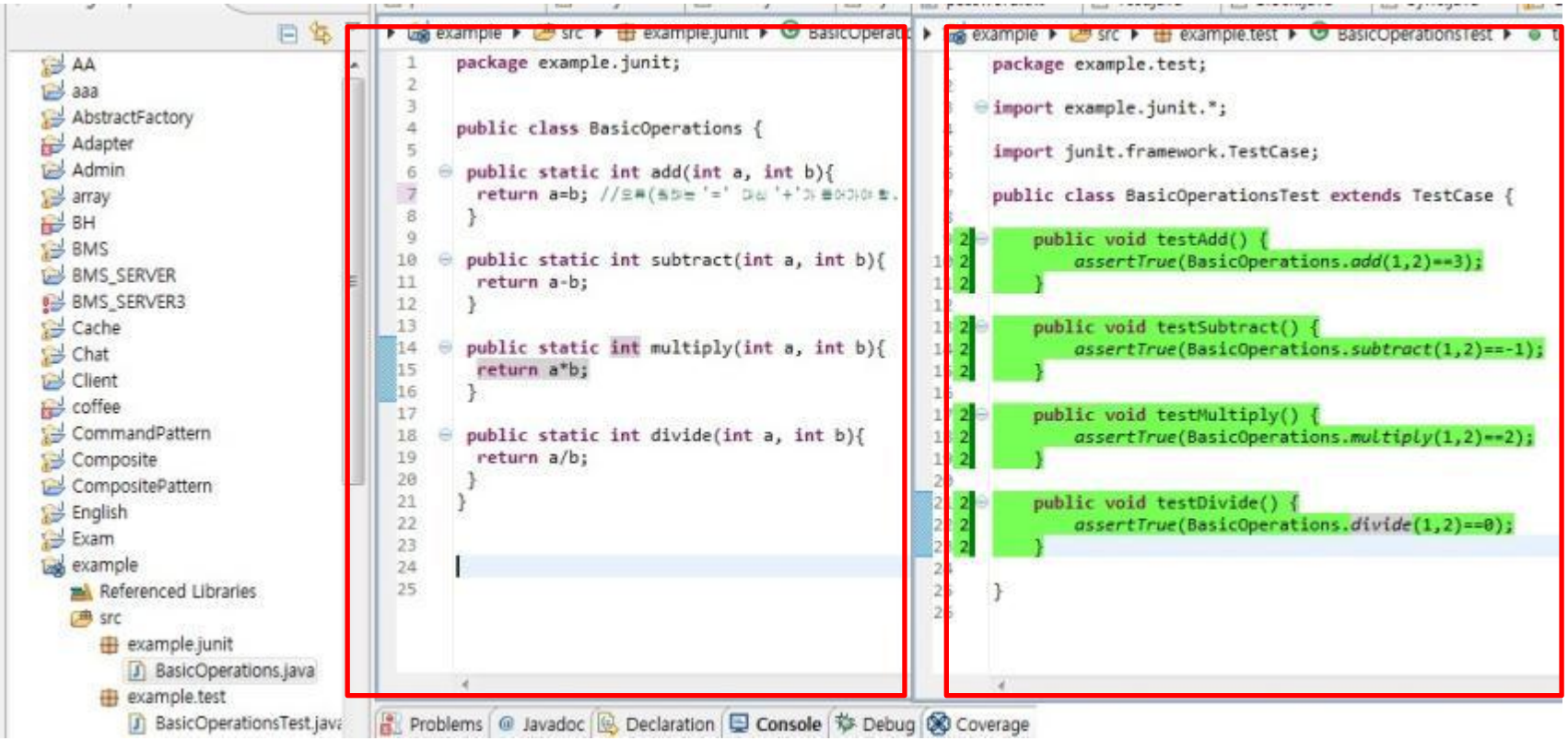
Clover 소개

» Clover 사용하기

» 2. 적용한 클래스 : BasicOperations

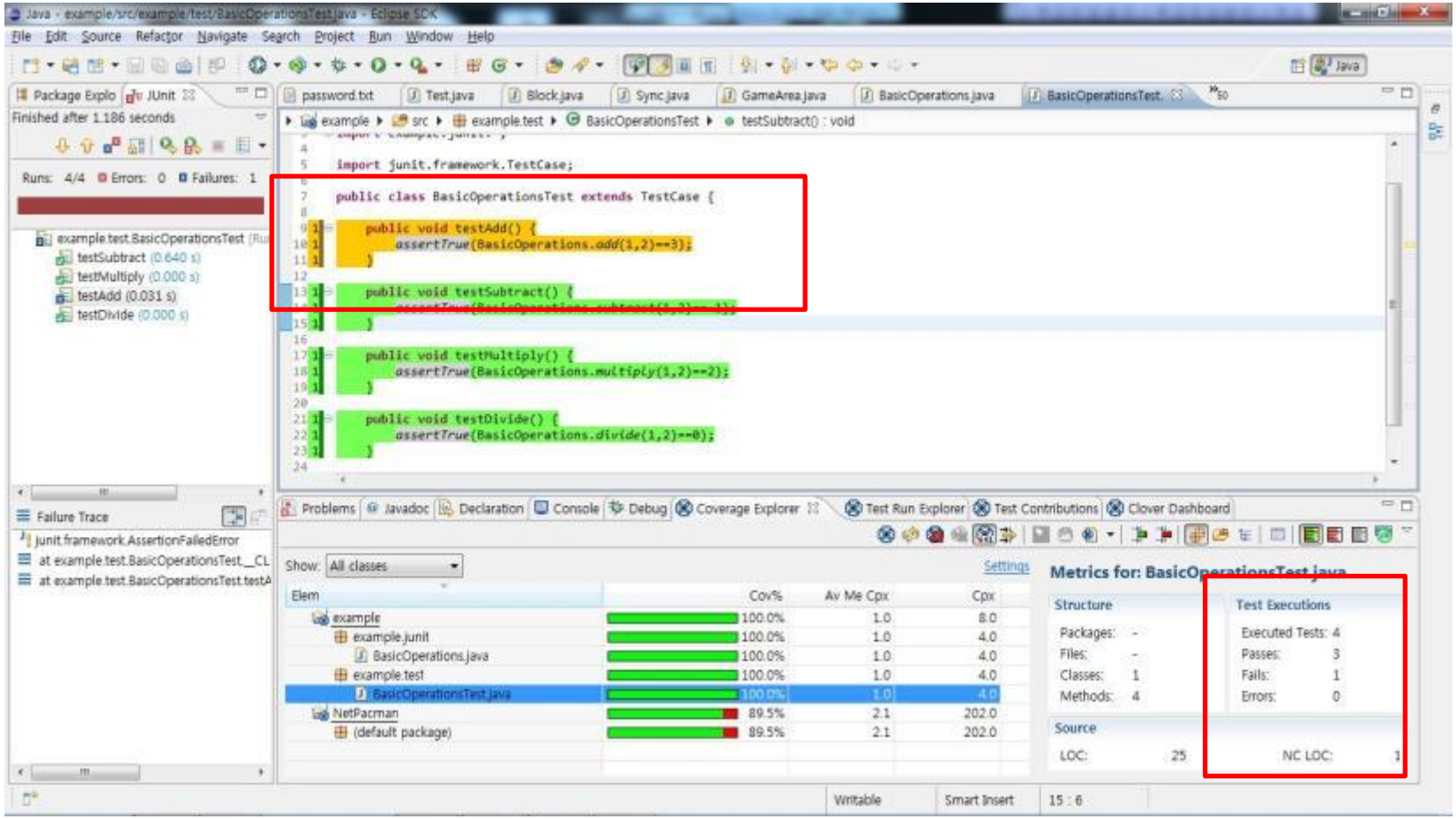
- +, -, *, / 연산을 수행하는 메소드 네 개가 있는 간단한

클래스와 이것을 테스트하는 테스트 케이스 클래스



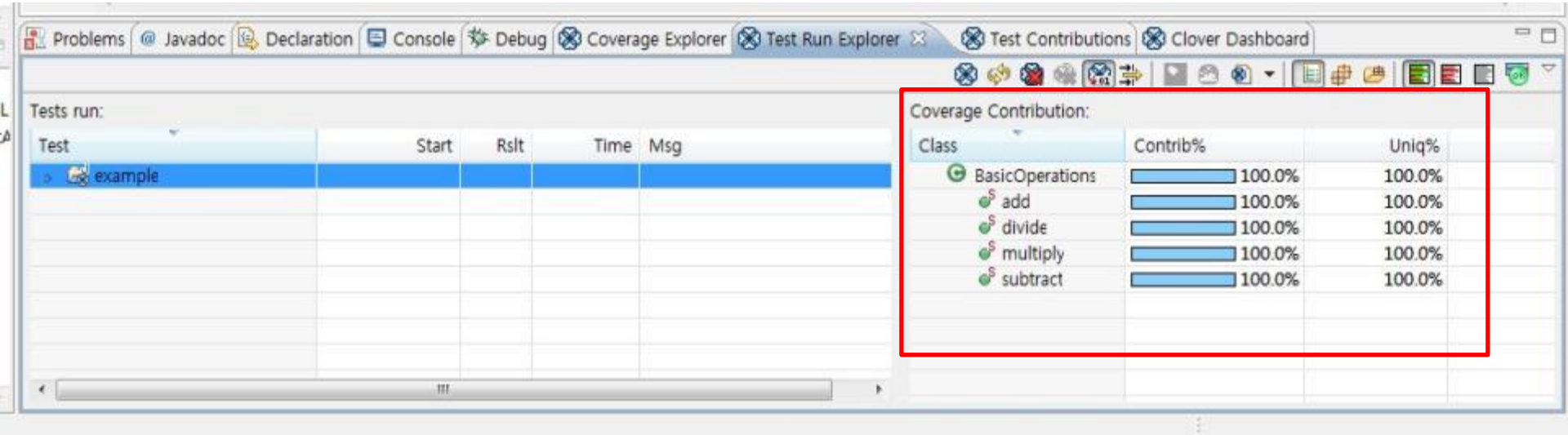
Clover 소개

» Coverage Explorer



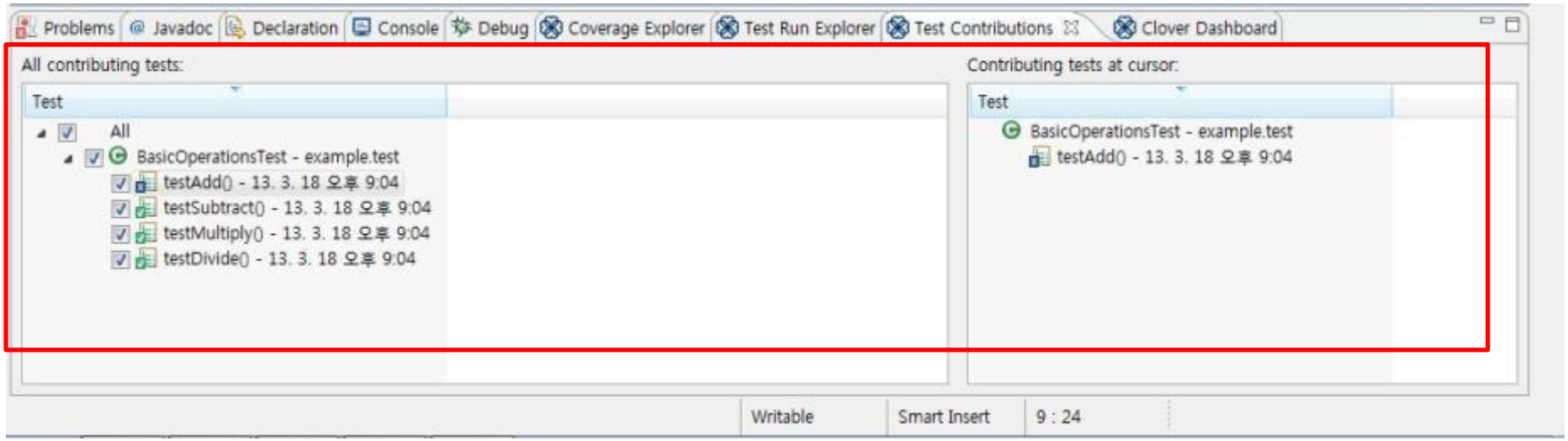
Clover 소개

» Test Run Explorer



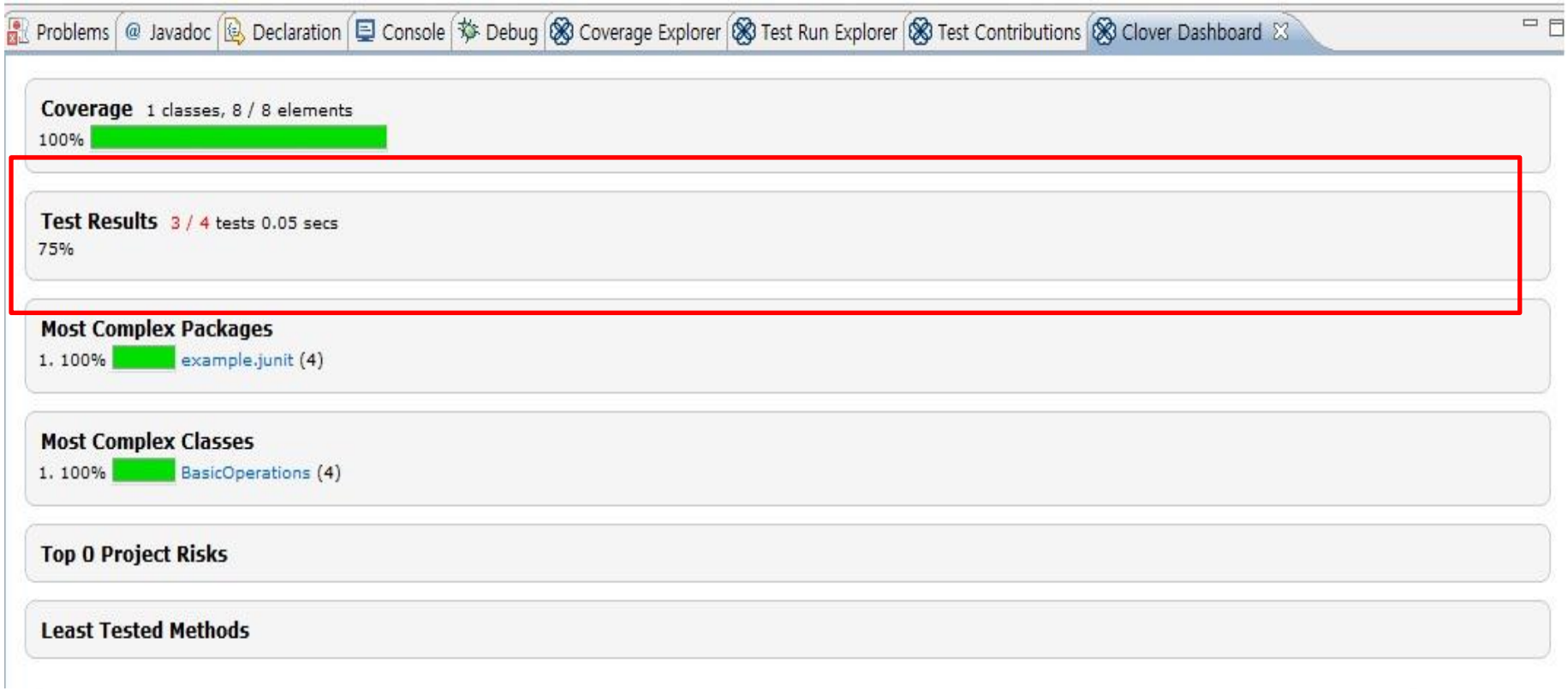
Clover 소개

» Test Contributions



Clover 소개

» Dashboard



Clover 사용 효과 & 어려웠던 점 (Code Coverage 적용 효과 & 어려웠던 점)

적용 시 좋은 점	적용 시 어려운 점
기존 소스 코드에서 리팩토링할 부분이 명확히 드러나고 좀 더 나은 설계가 가능	노하우와 지식 부족으로 다양한 경우에 대한 단위 테스트 작성이 힘듬
소스 코드를 수정할 때 코드 정상 동작 여부나 부작용 확인 가능	테스트 코드 작성에 많은 시간이 소요되어 개발 일정에 무리
객체에 대한 개별 검증 가능하여 전체 시스템이 구성되지 않아도 기능 확인이 쉬움	소스 코드를 변경할 때 테스트 코드도 함께 변경해야 함
실제 서비스 시에 드물게 발생하는 예외적 케이스에 대한 테스팅 가능	

의존성(Dependencies) 소개

의존성 소개

» 패키지 의존성(Package Dependencies)

- 한 패키지가 다른 패키지에 의존할 때 발생
- 아키텍쳐 안에서의 구조와 관계에 대해 판단 가능
- 다양한 시스템 테스트 및 모호한 런타임 오류를 처리할 필요 없이 end user에게 개발 상태를 확인할 수 있게 해주는 방법

의존성 소개

» 사이클 의존성(Cyclic Dependencies)

- 두 개 이상의 패키지가 서로 의존성을 가져 cycle을 이루는 상태
- 직접 사이클 의존성과 간접 사이클 의존성
- 프로그램의 안정성을 감소
- 정확한 시스템 변화의 영향 측정 및 관리를 방해

사이클 내에서 협동하는 패키지들
-> 원자적인 유닛으로 분리

JDepend 소개

JDepend 소개

JDepend란?

- 패키지에 대한 디자인 품질 측정 도구

JDepend 사용 이유

- 현재 패키지의 패키지 간 의존성 확인 가능
- 사이클 의존성 문제점 발견 가능
- 개발 도중에 문제점을 확인할 수 있어 용이
- 디자인을 측정하고, 그 디자인을 이해하고, 디자인이 계속적인 refactoring을 겪는 동안 예상된 품질을 나타내는지 자동적으로 체크

» 품질 측정 요소

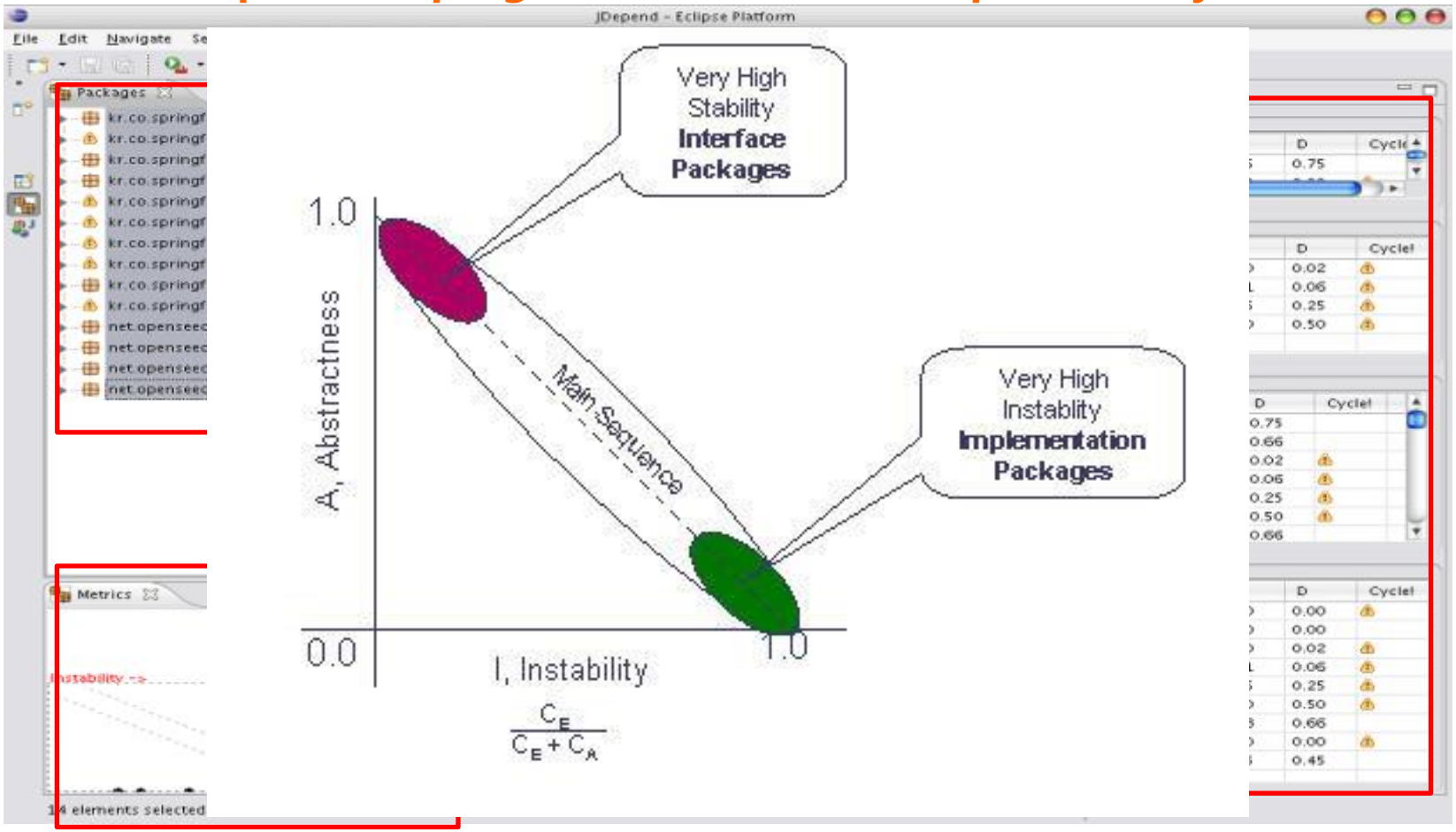
- 클래스와 인터페이스의 수 -> 패키지 확장 가능성 지표
- Ca (패키지 내의 클래스들에 의존하는 다른 패키지의 수)
  -> 패키지 응답성에 대한 지표
- Ce (패키지 내의 클래스들이 의존하는 다른 패키지들의 수)
  -> 패키지 독립성에 대한 지표
- A (모든 클래스에 대한 추상 클래스의 비율)
  -> concrete package / abstract package에 대한 지표
- I (Ce+Ca에 대한 Ce의 비율)
  -> 패키지의 변화에 대한 회복력(탄성)에 대한 지표
- D(하나의 패키지에서 이상적인 선의 수직 거리)
  -> abstractness와 stability 사이의 패키지 균형 지표
- 패키지 의존 사이클

» JDepend의 한계점

- 주어진 패키지의 모든 사이클을 보고할 수는 없다.
- 소스코드 complexity 측정치를 수집하지 않는다.
- 디자인 품질 측정치가 완벽하지는 않다.

» JDepend 사용 예

- JDepend 용 plug-in 설치 후 Run JDepend analysis 클릭



References

- **Junit**

http://ash84.tistory.com/772
http://javacan.tistory.com/179
http://www.agiledata.org/essays/tdd.html
http://en.wikipedia.org/wiki/Test-driven_development
http://blog.naver.com/ellay06?Redirect=Log&logNo=120170104099
http://blog.naver.com/ellay06/120169650178
http://lyb1495.tistory.com/73
http://blog.naver.com/palfuni?Redirect=Log&logNo=120154480776
http://cafe.naver.com/tech2u/1287
http://www.javajigi.net/pages/viewpage.action?pageId=278

- **Clover**

도서 : NHN은 이렇게 한다! 소프트웨어 품질관리
http://blog.naver.com/dive2net?Redirect=Log&logNo=40062353250
http://bcho.tistory.com/156
http://okjsp.tistory.com/1165642997
http://bullseye.com/coverage.html#basic_path
http://blog.daum.net/aqua0405/5558428
http://alica_park.blog.me/30120565913
http://www.sten.or.kr/bbs/board.php?bo_table=test_story

- **JDepend**

http://docs.fedoraproject.org/en-US/Fedora_Draft_Documentation/0.1/html/RPM_Guide/ch-dependencies.html
http://www.clarkware.com/software/JDepend.html
http://www.onjava.com/pub/a/onjava/2004/01/21/jdepend.html?page=1
http://blog.daum.net/darkandy/129

Thank you!