사례 01

C++언어로 개발된 게임 플랫폼의 테스트 자동화 사례

염선화·최홍석·전미숙 (NHN 게임QA실)

목 차 »

1. 서 론

테스트 단계의 생산성을 향상시키기 위해서 테스트 자동화는 널리 시도되고 있으며 다양한 자동화 도구가 소개되어 현업에서도 많이 도입하고 있다. 그러나 테스트 자동화는 초기 투입 비용의 증대를 포함하고 있기에 테스트 자동화를 도입하는 조직에서는 적합한 대상에 효율적으로 테스트 자동화를 추진하는 것이 중요하다.

NHN의 게임 포털인 한게임[1]에서는 게임 서비스의 공통 기능을 게임플랫폼으로 개발하여 적용하고 있다. 게임 플랫폼은 게임 서비스의 필수적인 기능을 수행하기 위해 다양한 서비스에 공통적으로 적용되기에 다양한 OS 및 분산 환경을 지원하며 지속적으로 업그레이드 된다. 이러한 특징으로 게임플랫폼의 검증을 위해서 반복적인 기능 테스트 및 높은 수준의 성능 테스트가 요구된다. 테스트 시간 및 비용을 절감하고 수동으로 수행하기 어려운 분산 환경 테스트를 효율적으로 수행하기 위해 C++언어로 개발된 게임 플랫폼의 테스트 자동화를 시도하였으며 본 글에서는 테스트 자동화 방안과 그 구현 사례를 소개한다.

2. 테스트 자동화의 방안

2.1 개발 및 통합 테스트 단계의 테스트 자동화 도구 단일화

테스트 자동화 구현을 위하여 다양한 자동화 도구 중 테스트 목적에 적합하다고 판단되는 자동화 도구를 선택하여 적용한다. 처음 게임플랫폼의 통합 테스트를 위한 테스트 자동화를 구현할 때 개발 단계의 테스트 구현 시 적용되고 있는 자동화 도구는 그다지 고려하지 않고 통합 테스트 단계에 적용하기 적합하다고 판단되는 테스트 자동화 도구를 별개로 선택하여 적용하였

다. 그 결과 테스트 자동화를 통해 테스트 수행은 효율적으로 수행할 수 있었으나 구현된 자동화 테스트를 수행하여 발견된 결함을 개발팀에서 확인하기 위해 통합 테스트 단계에서 사용된 별도의 자동화 도구를 설치해야 하거나 테스트의 동작을 이해하기 위해 추가적으로 자동화 도구의 사용법까지 숙지해야 하는 문제가 발생하였다. 따라서 개발 및 통합 테스트 자동화 구현 시 동일한 도구를 사용한다면 이러한 문제를 해결할 수 있게 되어 아래와 같은 효과를 기대할 수 있다.

- **동일한 도구 사용으로 개발과 QA간의 커뮤니케이션 비용 절감**

통합 테스트 단계에서 결함이 발견되면 QA는 결함 현상에 대한 설명과 결함을 발견하기 위해 수행한 테스트 내용을 개발팀에 전달하여 결함의 원인 규명 및 수정을 요청한다. 그러나 동일한 결함 내용에 대해 종종 개발과 QA간 다르게 이해하는 경우도 생기며 발견된 결함의 재현이 쉽지 않아 어려움을 겪기도 한다. 개발과 QA간 동일한 자동화 도구를 사용하고 있다면 부가의 작업이나 설명 없이 자동화 테스트를 전달하고 수행하는 것만으로도 발견된 결함에 대한 명확한 설명 및 재현이 가능하여 커뮤니케이션 비용의 절감을 가져올 수 있다.

- **자동화 테스트의 공유로 빠른 시점의 테스트가 가능**

통합 테스트 단계에서 개발되는 자동화 테스트를 개발 단계에서 공유하여 수행한다면 결함의 조기 발견을 가능케 하는 좋은 방법이 될 수 있다. 개발단계와 통합 테스트 단계의 테스트 자동화 도구를 동일하게 가져간다면 개발 단계에서 추가적인 구현이나 환경 설정 없이 QA가 작성한 테스트를 수행하고 결과를 확인할 수 있어 통합 테스트 단계 이전에 개발 산출물의 검증이 가능하다.

- **유닛 테스트 코드 재활용으로 통합 테스트 자동화 비용 절감**

게임플랫폼의 통합 테스트를 자동화 하기 위해서는 기능 명세서나 상세 설계서를 기반으로 테스트 케이스를 추출하고 이에 맞게 테스트 코드를 구현하여 자동화한다. 이때 작성되는 테스트 코드를 살펴본 결과 플랫폼이 외부 모듈에 서비스나 기능을 제공하기 위해 노출하는 API를 다양하게 호출하는 방식으로 많이 구현되었다. 따라서 동일한 테스트 자동화 도구 상에서 작성된 API의 유닛 테스트 코드는 프로그래밍 예제 역할을 하여 통합 테스트 코드 작성 시 이해를 돕고 입력되는 테스트 데이터만 변경하는 방식으로 코드 재활용이 가능하여 보다 손쉽게 통합 테스트를 자동화 할 수 있다.

- **유닛 테스트와 통합 테스트 연계를 통한 테스트 커버리지 향상**

개발 단계에서 작성된 유닛 테스트 코드를 공유 받아 통합 테스트 자동화를 작성하게 되면 유닛 테스트 단계에서 테스트가 다소 부족하다고 판단되는 모듈의 테스트를 통합 테스트 시 좀더 집중 수행하여 전체적인 테스트 커버리지를 향상시킬 수 있다.

2.2 분산 환경의 End to End 테스트 자동화

게임 서비스의 공통적인 기능을 제공하는 게임 플랫폼의 경우 적용되는 게임 서비스의 규모에 따라 여러 대의 서버에 적용되어 작동하게 된다.

(그림 1) End to End 테스트 자동화 작업 순서도

이러한 플랫폼의 성능 테스트를 수행하기 위해서는 다수의 테스트 서버에 테스트 환경을 구성하고 자동화 테스트를 배포 실행하여 테스트 결과를 분석해야 하는데 수동으로 일련의 테스트 작업을 반복 구성하고 실행하는 것은 많은 시간이 소요되며 분산된 환경에서 동시에 테스트가 수행되어야 하는 경우는 테스트가 아예 불가능해진다. 따라서 분산 환경에서의 효과적인 테스트를 위해서는 End to End 테스트 자동화가 요구된다.

End to End 테스트 자동화는 (그림 1)과 같이 "환경 구성", "테스트 수행", "결과 분석", "환경 제거"의 일련의 단계를 포함해야 한다. 테스트 환경 구성은 테스트를 수행하기 위하여 SUT (System Under Test)와 SUT가 사용하는 DOC (Depended On Component)를 테스트 장비에 설치하고 초기 상태로 설정하는 단계를 말한다. 환경 구성이 끝나면 자동화 테스트를 원격으로 수행시키고 그 결과를 분석 장비로 취합하고 최종적으로 테스트 장비에 설치하거나 기록하였던 환경 및 데이터를 제거해야 한다. [2]

게임 플랫폼의 분산 테스트를 손쉽게 가능하게 하기 위해서 End to End 테스트 자동화를 위한 일련의 작업을 순서대로 수행하는 도구를 제작하였으며 도구의 구현 내용은 3.2장에서 소개한다.

3. 테스트 자동화의 구현

3.1 Google Test를 이용한 테스트 자동화 구현

2.1장에서 설명한 바와 같이 개발단계에서부터 QA단계의 테스트까지 동일한 테스트 프레임워크를 사용하는 것은 테스트의 원활한 공유 및 단계별 테스트의 연계를 가능케 하여 좀더 낮은 비용으로 좋은 테스트 효과를 기대할 수 있다. 따라서 개발단계에서 유닛 테스트 프레임워크로 활발히 사용되고 있는 Google Test[3]를 이용하여 통합 테스트 자동화를 구현하였다. 다양한 C++ 테스트 자동화 도구 중 다음과 같은 장점으로 Google Test를 선택 적용하였다.

- **Learning Curve가 낮음** : C++에 대한 기본적인 개발 능력이 있으면 Google Test에서 제공하는 가이드 문서만 참고하여 쉽게 사용이 가능하다.
- **디버깅(Debugging)이 쉬움** : Google Test는 IDE(Integrated Development Environment)에서 개발하는 일반적인 방식과 동일하게 디버깅이 가능하다.
- **테스트 수행이 간단** : 추가적인 환경 구성 없이 컴파일 된 실행 파일만 가지고 테스트 수행이 가능하다.

- **테스트 수행 결과를 XML 형식으로 리포트** : XML 파일로 결과를 생성해 주기 때문에 결과 가공 및 타 시스템 연동이 편리하다.
- **선택적인 테스트 수행** : '-gtest filter' 옵션을 통해 특정 테스트만 수행 가능하다. 예를 들어 '-gtest filter=Footest.*' 옵션의 경우 테스트 케이스가 'Footest'인 모든 테스트를 수행한다. 또한 '-gtest filter=Foo*." 옵션은 테스트 케이스가 'Foo'로 시작하는 모든 테스트를 수행할 수 있어 원하는 테스트만 선택하기가 쉽다.

그러나 Google Test가 유닛 테스트 자동화에 널리 사용되는 도구이다 보니 통합 테스트 자동화를 구현하기에 몇 가지 아쉬운 점이 있었으며 다음과 같은 보완책을 구현하여 적용하였다.

- **테스트 데이터의 외부 입력**

Google Test를 이용하여 테스트를 구현 시 테스트 데이터가 테스트 코드와 분리되지 않고 직접 입력하여 코드를 구현하게 된다. 이러한 방식은 다양한 테스트 데이터를 입력하여 테스트를 수행해야 하는 통합 테스트에 적용하기에는 테스트 코드 작성이 번거롭고 테스트의 유지 보수성도 저하시킨다. 또한 테스트 데이터 변경 시마다 매번 다시 빌드를 해야 하기에 테스트 생산성도 떨어지는 문제점이 있다. 이러한 점을 보완하기 위해서 XML로 작성된 별도의 테스트 데이터를 테스트 수행 시 동적으로 읽어 들여 테스트를 수

```
Int main(int argc, char **argv)
{
    ::testing::InitGoogleTest(&argc,argv);
    Bool result = RUN_ALL_TESTS();
    Return result;
}

TEST(FactorialTest, HandlesPositiveInput)
{
    EXPECT_EQ(1, Factorial(1));
    EXPECT_EQ(2, Factorial(2));
    EXPECT_EQ(3, Factorial(3));
    EXPECT_EQ(40320, Factorial(8));
}
```

(그림 2) Google Test만 사용하여 구현된 테스트 코드의 예

```
Int main(int argc, char **argv)
{
    LoadDataFromXML("testdata.xml");
    ::testing::InitGoogleTest(&argc,argv);
    Bool result = RUN_ALL_TESTS();
    Return result;
}

TEST(FactorialTest, HandlesPositiveInput)
{
    N_EXPECT_EQ("expect", Factorial("input"));
    // 'N_EXPECT_EQ'는 gtest의 'EXPECT_EQ'함수를 Wrapping하여 테스트 데이터 개수만큼
Iteration을 수행하는 함수(or MACRO)
```

testdata.xml

```
<?xml version = "1.0"?>
<testdata>
    <testcase name="FactorialTest">
        <test name="HandlesPostiveInput">
          <data input="1" expect="1" />
          <data input="2" expect="2" />
          <data input="3" expect="6" />
          <data input="8" expect="40320" />
        </test>
    <test name="OtherTest">
      ……
    </test>
  </testcase>
  <testcase name="OtherTestCase">

  </testcase>
  ……
</testdata>
```

(그림 3) 테스트 데이터 XML Mapping 라이브러리를 사용하여 구현된 테스트 코드의 예

행하는 '테스트 데이터 XML Mapping 라이브러리'를 제작하여 적용하였다. 이를 이용하면 테스트 데이터 수정 및 추가 시 별도의 코드 빌드 없이 XML 파일만 변경하여 테스트가 가능하다.

· 원격 호스트(Host) 제어

Google Test는 자체적으로 외부에 있는 호스트 제어 기능을 가지고 있지 않다. 간단하게 원격으로 호스트를 제어하고 싶은 경우 STAF[4]이나 다른 도구를 이용하여 외부 호스트를 제어하는 스크립트나 실행 파일을 작성하고 테스트 코드에서 System Call 함수(exec함수)를 통해 원하는 시점에 해당 스크립트나 실행파일을 호출하는 방법으로 (그림 4)와 같이 사용이 가능하다.

이 방법을 사용하는 경우 단순하게 외부에서 작성된 내용을 실행만 해주기 때문에 결과값 확인 및 추가적인 작업에는 한계가 있다. 따라서 추가 기능이 필요한 경우에는 STAF 라이브러리를 테스트 코드에서 직접 사용하거나, 사용에 대한 요구가 많다면 기본적인 원격 호스트 제어 기능을 가진 라이브러리를 구현하여 앞에서 설명한 '테스트 데이터 XML Mapping 라이브러리'와 함께 배포하는 방법도 고려해 볼 수 있다.

· 문서와 테스트 코드의 분리

Google Test는 소스코드 이외에 별도로 문서를 정리해 놓을 수 있는 방법이 없다. Doxygen 형식의 주석에 대한 표준을 정하여 테스트 코드에 주석을 남기도록 하여, Doxygen을 이용하여 별도의 문서를 생성해 낸다면 보완이 가능하다.

· 시나리오 테스트

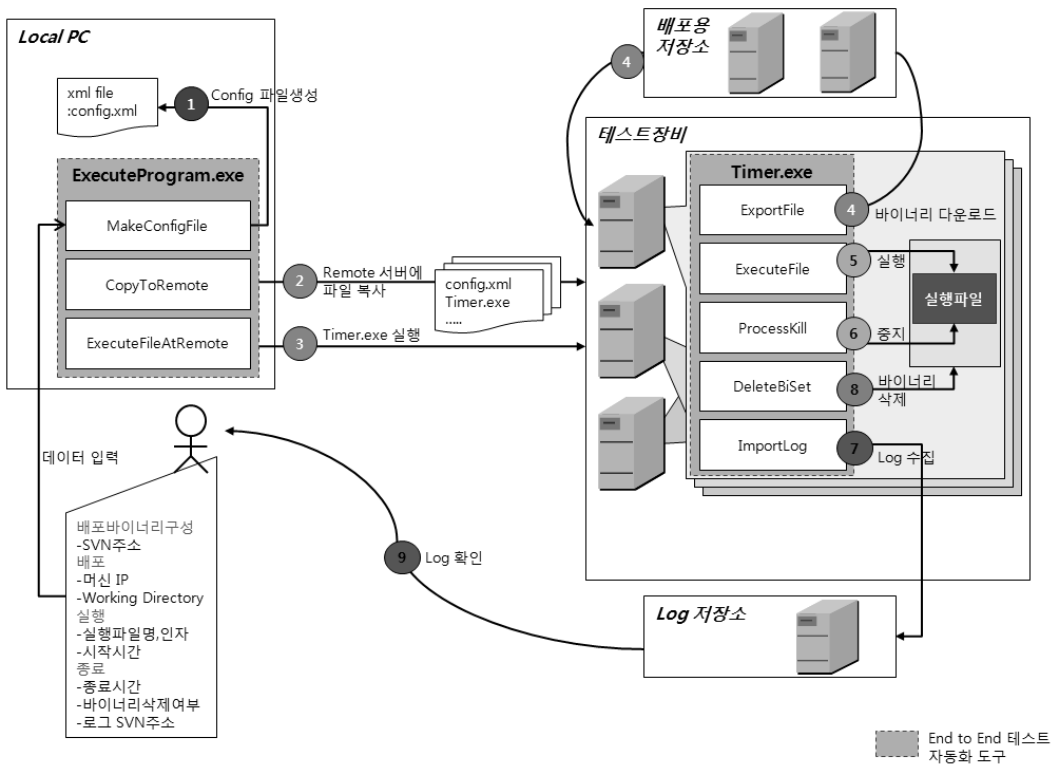
Google Test는 동적인 테스트 시나리오 구성이 어렵기에 테스트 시나리오를 코드로 구현해야 한다. 따라서 시나리오 변경 시 테스트 코드의 재빌드가 필요하다. 현재 게임플랫폼 통합 테스트의 테스트 시나리오의 많은 부분이 다소 간단하게 구성되어 있어 Google Test로 구현하여도 무리가 없으나 복잡한 시나리오 테스트가 요구되는 경우 한계가 있다. 복잡한 테스트 시나리오의 자동화를 위해서는 다른 도구를 도입하거나 다른 보완책이 필요하므로 이러한 테스트가 요구가 많은 경우는 Google Test의 도입을 신중히 고려해야 한다.

3.2 테스트 배포 및 실행 자동화 구현

분산된 한대 이상의 서버에서 테스트를 수행하여야 하는 경우가 많은 게임 플랫폼의 테스트를 위해 테스트 수행을 위한 일련의 단계를 자동으로 수행할 수 있도록 하는 End to End 테스트 자동화 도구를 구현하였다. 테스트 환경 구성에 필요한 바이너리 구성부터 배포, 실행, 로그 수집, 환경 제거를 하나의 도구 상에서 손쉽게 자동으로 수행할 수 있는 UI(User Interface)를 가진 도구로서 동작 방식은 (그림 5)와 같다.

```
Int main(int argc, char **argv)
{
    ::testing::InitGoogleTest(&argc,argv);
    Exec("setupscript.exe")
    Bool result = RUN_ALL_TESTS();
    Exec("teardownscript.exe")
    Return result;
}
```

(그림 4) 원격 호스트 제어 테스트 코드의 예



(그림 5) End to End 테스트 자동화 도구

4. 테스트 자동화의 효과

일부 게임 플랫폼의 통합 테스트를 3.1장에서 제시된 보완책과 함께 Google Test를 이용하여 자동화하였다. 통신 프로토콜 개발에 적용되는 라이브러리 성 게임 플랫폼을 대상으로 Google Test로 자동화 한 유닛 테스트 코드를 개발팀으로부터 공유 받아 통합 테스트 코드 작성시 재활용을 시도하였으며 모듈 별 유닛 테스트 커버리지를 미리 분석하여 커버리지가 낮은 모듈의 통합 테스트를 좀더 보강하는 방식을 취하였다. 그 결과 <표 1>과 같이 통합 테스트 자동화에 투입되는 리소스가 유닛 테스트 코드를 재활용하지 않는 경우에 비해서 다소 감소하는 효과를 얻었으며 유닛 테스트 및 통합 테스트의 총 테스트 커버리지도 전체적으로 상승하였다. 다만, 유닛 테

〈표 1〉 유닛 테스트 코드 재활용의 효과 – 통신 프로토콜 개발 플랫폼의 적용 예

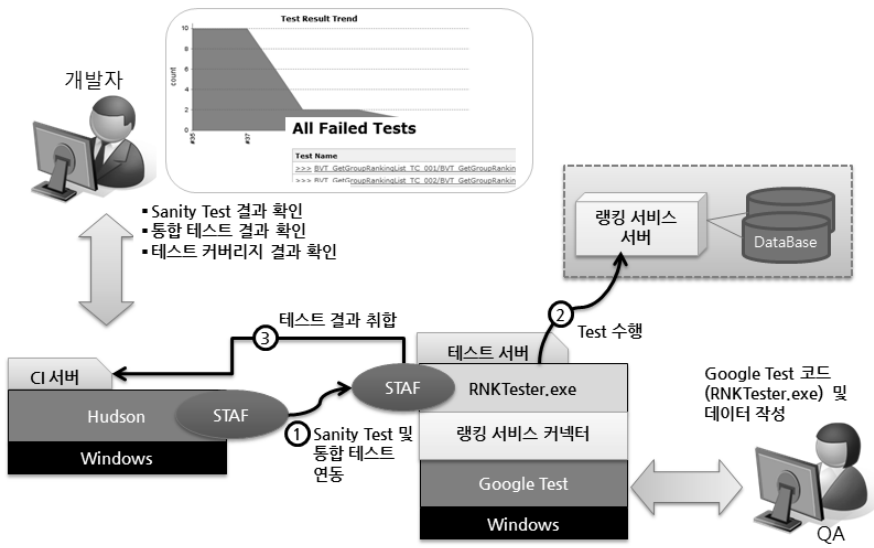
[투입 리소스 비교]

	재활용하지 않은 모듈	재활용한 모듈
TC* 수	36	38
자동화 구현 투입 리소스	48 Hours	32 Hours
TC당 구현 투입 리소스	1.33 Hours	0.84 Hours

[테스트 커버리지(Branch Coverage) 추이]

유닛 테스트 커버리지	통합 테스트 커버리지	총 커버리지
66 %	52 %	70 %

*TC : Test Case



(그림 6) CI서버를 통한 자동화 테스트의 공유 – 랭킹 서비스 플랫폼의 적용 예

스트 코드를 재활용하기 위해서는 유닛 테스트 코드 상에 테스트의 명세가 있는 경우 좀더 용이하며 통합 테스트 케이스가 주로 API 검증에 집중되는 라이브러리 성 플랫폼일 경우 좋은 효과를 기대할 수 있었다.

또한 Google Test를 이용하여 통합 테스트를 위해 QA가 작성한 테스트 중 주요 기능의 정상 동작을 테스트 하는 자동화 테스트를 선별하여 통합 테스트 시작 전 개발에 전달하였다. 개발과 동일한 테스트 자동화 도구를 사용하였기에 공유된 테스트는 추가적인 환경 세팅 없이 손쉽게 개발팀에서 데일리 빌드를 수행하는 CI(Continuous Integration) 서버에 연동되었다. 연동된 테스트는 개발 산출물의 품질을 선 검증하는 Sanity Test로서 사용되어 주요 결함을 좀더 빠른 시점에 검출하는 좋은 장치가 되었다. 또한 자동화된 통합 테스트도 CI 서버에 연동하여 수행된 통합 테스트의 결과 및 어떤 코드에서 테스트가 실패하였는지 한눈에 확인이 가능하였다.

구현된 End to End 자동화 도구는 오픈 마켓 게임 서버 플랫폼의 성능 테스트에 적용하였다. 특정한 로직을 수행하도록 구현된 자동화 테스트를 30대의 테스트 머신에 배포 및 실행하고 테스트 로그를 회수하는 시나리오의 테스트이며 도구 적용 전, 후의 효과는 <표 2>와 같다.

〈표 2〉 End to End 자동화 도구 적용 전,후 효과 비교 – 게임 서버 플랫폼의 성능 테스트 적용 예

	적용 전	적용 후
테스트 방법	STAF 만 이용	End to End 자동화 도구 이용
배포 소요 시간	머신 한대당 약 40분 소요 순차적 배포로 총 (40*30)분 소요	머신 한대당 약 5분 소요 동시 배포로 총 5분 소요됨
테스트 실행 / 중지 방법	직접 머신마다 접속하여 실행하고 테스트 종료 확인 후 중지	설정된 시작에 테스트 자동 실행/중지
결과 취합	직접 머신에 접속하여 로그 파일 회수	자동으로 로그 저장소에 저장

5. 결 론

반복적인 기능 테스트 및 높은 수준의 성능 테스트가 요구되는 게임 플랫폼의 테스트를 효과적으로 수행하기 위해서 테스트 자동화는 좋은 수단이다. 그러나 자동화를 구현하는 것은 수동 테스트에 비해 자동화 비용이 추가된다. 따라서 추가 비용에 비해 얻을 수 있는 효과를 높이기 위해서 전략적으로 자동화를 시도해야 한다.

본 글에서는 게임 플랫폼의 효과적인 테스트를 위하여 개발팀에서 많이 사용하는 자동화 도구인 Google Test를 이용한 통합 테스트 자동화 및 End to End 자동화의 방안과 그 구현 사례를 소개하였다. 이러한 자동화 방안을 적용하여 테스트 수행 시간을 단축하는 것은 물론이고 개발과 QA간의 커뮤니케이션 비용 감소 등 테스트 생산성 향상 효과와 QA 단계 이전의 빠른 개발산출물 검증도 가능하였다.

참 고 문 헌

[1] "한게임", http://www.hangame.com

[2] Gerard Meszaros, xUnit Test Patterns, Addison Wesley, 2007

[3] "Google Test", http://code.google.com/p/googletest

[4] "Software Testing Automation Framework (STAF)", http://staf.sourceforge.net

저 자 약 력

염 선 화

이메일 : rangenabi@naver.com

- 1999년 이화여자대학교 컴퓨터공학과(학사)
- 2001년 이화여자대학교 컴퓨터공학과(석사)
- 2007년~현재 NHN 게임QA실 / 차장
- 관심분야 : 테스트 자동화, 모바일 테스트, 임베디드 소프트웨어

최 홍 석

이메일 : pivot@naver.com

- 2001년 중앙대학교 전자전기공학부(학사)
- 2007년~현재 NHN 게임QA실 / 차장
- 관심분야 : 테스트 자동화, 모바일 테스트, 웹 프로그래밍

전 미 숙

이메일 : mymelody01@naver.com

- 2003년 경북대학교 통계학과(학사)
- 2007년~현재 NHN 게임QA실/과장
- 관심분야 : 테스트 자동화, 모바일 프로그래밍