설계명세서

제 10회 공개 SW 개발자 대회

스펙트럼 기반 통합 테스트 플랫폼

SWQ&R

성균관대학교
소프트웨어공학연구실
김정호
jeonghodot@skku.edu
010-3111-3555

1. Preface

1.1. Objective

Preface에서는 본 문서의 독자가 누구인지를 밝히고 문서의 구조와 각 챕터의 역할을 소개한다. 또, Version History를 제시하여 문서의 새로운 버전이 만들어질 때마다 변경사항과 변경사유를 설명한다.

1.2. Readership

본 문서의 독자는 다음과 같다. 시스템을 직접 개발하는 소프트웨어 엔지니어, 시스템을 설계하는 아키텍처와 개발에 참여하는 모든 구성원을 독자로 정의한다. 만약, 시스템을 개발할 때 외주 업체를 이용한다면, 해당 업체에서 개발에 관련되는 모든 구성원 역시 독자에 포함한다. 즉, 본 문서의 독자는 본 문서에서 소개하는 시스템의 개발 및 유지 보수에 관련된 모든 구성원이다.

1.3. Document Structure

이 문서는 총 10개의 부분으로 구성되어 있다. Preface, Introduction, Glossary, User Requirements Definition, System Architecture, System Requirements Specification, System Models, System Evolution, Appendices, Index 로 구성된다. 각 부분의 역할은 다음과 같다.

1.3.1 Preface

Preface에서는 본 문서의 예상 독자가 누구인지 밝히고, 문서의 구조, 각 챕터의 역할 등에 대해 소개한다. 또, Version History와 그 변경사유, 변경사항에 대해 설명하였다.

1.3.2. Introduction

Introduction에서는 본 문서에서 시스템의 설계할 때 사용하는 모든 종류의 다이어그램 및 툴에 관해 서술한다.

1.3.3. System architecture

System Architecture에서는 우리 팀에서 개발하고자 하는 시스템에 대해 전반적으로 서술한다. 시스템의 전체적인 구조를 설명한다. 또, 시스템을 Block diagram으로 나타내어, 각각의 관계와 실제 어떻게 사용되는 지를 Package diagram과 Deployment diagram으로 사용하여 설명한다. 각 시스템에 대한 modular decomposition은 4~9장에 걸쳐 설명한다.

1.3.4. Test Case Prioritizater

사용자의 결함 추적을 위한 시간을 줄이기 위한 테스트 케이스 우선순위화 시스템의 설계를 설명한다. Class diagram, sequence diagram, 그리고 state diagram을 통하여 세부 컴포넌트를 표현하고 이에 대하여 설명한다.

1.3.5. Fault Localizer

ISES 내부에서 동작하는 소스코드 및 테스트 케이스 자동 수행을 위한 시스템 설계를 설명한다. Class diagram, sequence diagram, 그리고 state diagram을 통하여 세부 컴포넌트를 표현하고 이에 대하여 설명한다

1.3.6. Input/Output System

테스트 케이스를 통해 결함 추적을 하고자 하는 사용자가 결함 추적을 위한 파일과 관련 테스트 케이스를 등록하고, 추적 과정과 결과를 파일로 저장하기 위한 파일 입출력을 위한 시스템의 설계를 설명한다. Class diagram, sequence diagram, 그리고 state diagram을 통하여 세부 컴포넌트를 표현하고 이에 대하여 설명한다.

1.3.7. User Interface Design

User Interface Design에서는 사용자에게 제공될 화면에 대해 설계하고 개별 기능에 대해 설명한다.

1.3.8. Testing Plan

시스템이 의도한 방향으로 실행되고 시스템 내부의 결함을 찾기 위해 testing을 한다. 이를 위해 설계단계에 미리 계획한다. 이 때 Testing Plan에서는 Testing Policy와 여러 Test Case에 대해 기술한다

1.3.9. Development Environment

Development Environment에서는 개발자의 환경에 대해 설명한다. 사용한 프로그래밍 언어와 IDE에 대해 서술한다.

1.3.10 Index

Index 에서는 주요 용어, 다이어그램, 기능에 대한 인덱스 등이 포함된다.

1.4. Version of the Document

1.4.1. Version format

버전 번호는 major.minor[.maintenance]로 구성되며. 문서의 버전은 0.1부터 시작한다.

1.4.2. Version management policy

설계 명세서를 수정할 때 마다 버전을 업데이트한다. 다만 변경간의 간격이 1 시간 이내 일 때에는 버전 번호를 업데이트 하지 않고 하나의 업데이트로 간주한다. 이미 완성된 파트를 변경할 때에는 minor number를 변경하며, 새로운 부분을 추가하거나 문서의 구성이 예전에 비해 괄목할 변화가 있을 경우 major number를 변경한다. 이미 작성한 부분에 대해서 오타 수정하거나, 문서의 구조를 변경할 경우 maintenance number를 추가하거나 변경한다.

1.4.3. Version update history

Version	Modified date	Explanation
0.1	2016-08-30	문서의 초안과 표지 작성
0.2	2016-08-31	Introduction 작성
0.3	2016-09-07	System Architecture 작성
0.4	2016-09-08	Test Case Prioritizater 작성
0.5	2016-09-09	Fault Localizer 작성
0.6	2016-09-10	Input/Output System 작성
0.7	2016-09-11	User Interface Design 작성
0.8	2016-09-12	Testing plan 작성
0.9	2016-09-12	development environment 작성
1.0	2016-09-13	전체 작성 완료

2. Introduction

2.1. Objectives

Introduction에서는 본 문서에서 시스템의 설계할 때 사용하는 모든 종류의 다이어그램 및 툴에 관해 서술한다.

2.2. Applied Diagram

2.2.1. UML

소프트웨어 시스템이 과거에 비해 점점 거대해 지고 복잡하게 되면서 좀 더 정확한 프로그래밍과 효과 있는 유지/보수가 요구 되고 있는 실정이다. 그리고, 이를 위해 소프트웨어의 모델링(Modeling)에 대한 개념이 아주 중요하게 되었다. 이러한 모델링을 위해 문자(Text)가 아닌 그림적(Graphical)인 요소를 사용하여 시스템을 표현핛 때 우리는 이것을 "비주얼 모델링(Visual

Modeling)"이라 하는데, 이를 시스템의 모든 stakeholder들이 동일하게 이해하도록 하기 위해서는 그래픽적인 표기법이 표준화할 필요가 있었다.

객체지향 시스템 개발 단체가 모인 OMG(Object Management Group)는 객체지향 모델링 언어의 표준을 정하고자 했고, 객체지향 패러다임 진영에 있던 Ivar Jacobson, Grady Booch, James Rumbaugh 이 세 사람이 기타 단체와 더불어 표준 모델링 언어를 OMG에 제안하면서, OGM가 표준으로 채택하여 만들어진 것이 바로 UML이다.

표 1 UML 2.0의 13가지 다이어그램

모델	다이어그램 계층	다이어그램	표현하는 내용
정적 모델	구조 다이어그램	클래스(Class)	클래스 구조
		객체(Object)	인스턴스 구조
		컴포넌트(Component)	시스템을 구성하는 컴포넌트들의 구조
		컴포지트(Composite)	시스템을 실행할 때의 구조를 나타내는 컴포지트들의 구조
		패키지(Package)	내부에 모델 요소를 포함할 수 있는 패키지 구성
		배치(Deployment)	시스템의 물리적 하드웨어 구성
동적 모델	상호작용 다이어그램	시퀀스(Sequence)	오브젝트 사이의 메시지 교환
		타이밍(Timing)	한 상태에서 객체가 지체하는 시간
		통신(Communication)	객체 사이의 메시지 교환
	상호작용 + 상호작용 내 행위	인터액션(Interaction)	객체 간의 메시지 교환을 액티비티 다이어그램과 같은 형태로 표현
	상호작용 내 행위	액티비티(Activity)	액티비티의 실행 순서, 조건, 실행자의 관계
		상태전이 (State-machine)	상태전이와 상태전이에 따른 행동
요구 분석		유즈케이스 (Use-case)	시스템이 제공할 서비스와 그 사용자의 관계

United Modeling Language(이하, UML)은 객체지향 소프트웨어 설계를 위해 사용되던 여러 종류의 다이어그램들을 통합하여 만든 모델링 표기법으로, 현재 객체지향 시스템 개발 분야에서 가장 우수한 모델링 언어로 인식되고 있다. 그 이유는 UML이 개발자가 개발하려고 하는 소프트웨어를 구현하기에 앞서서 표준화되고 일반화되는 방식으로 설계되어 소프트웨어에 관련된 모든 사람들과의 상호 소통을 해줄 수 있는 메커니즘이 되기 때문이다.

UML은 개발할 시스템의 유형에 관계 없이 모든 시스템에 대해 적용될 수 있다. 또한 시스템 개발 방법론, 개발에 사용되는 프로그래밍 언어, CASE 도구에 관계 없이 적용될 수 있는 일반적인 표현 방법이기 때문에 UML을 이용하여 시스템 설계를 하였다.

위에서 소개한 UML은 객체 지향 설계를 위한 표준 언어로서, 소프트웨어 개발의 산출물을 시각화, 상세화, 구축, 문서화하는 특징을 가지고 있다. 이 4가지 특징에 대해 자세하게 설명한다.

1) UML은 시각화, 가시화 언어이다.

UML은 소프트웨어 개념 모델을 시각적인 그래픽 형태로 작성하며, 그 표기법에 있어서는 각 심볼에 대한 명확한 정의가 존재한다. 따라서 개발자들 사이에 오해 없는 원활한 의사소통이 이루어질 수 있으며, 여러 가지 시각적인 다이어그램을 통해 특정 관점으로 시스템을 표현할 수 있다.

2) UML은 상세화, 명세화 언어이다.

UML은 시스템 개발 과정인 분석, 설계, 구현 단계의 각 과정에서 필요한 모델을 정확하고 명백하고 완전하게 표현할 수 있는 수단이다. UML을 통해 시스템 설계의 복잡성을 명확하게 기술한다.

3) UML은 구축 언어이다.

UML은 C++, Visual Basic, Java 등과 같은 다양한 프로그래밍 언어로 표현할 수 있다. 따라서 UML로 상세화 된 설계 모델은 프로그래밍 소스 코드로 변환하여 구현이 가능하다. 또한 이미 구현되어 있는 소스 코드를 UML로 역변환하는 데 이용할 수 있어서 분석하는 역공학(Reverse Engineering)이 가능하다.

2.2.2. Class Diagrams

클래스 다이어그램은 "클래스"라고 하는 속성(Attribute)과 메소드(Method)를 포함하고 있는 객체지향 설계단위를 이용해 시스템의 주로 정적인 구조를 표현하는 다이어그램이다. 박스 안에 3개의 부분에는 위에서부터 아래로 각각 클래스의 이름, 클래스의 속성, 클래스의 메소드가 들어가게 된다. 클래스 다이어그램은 객체지향 모델링에서 특히 많이 사용되는 개념으로, 클래스로부터 상속이나 기타 관계를 통해 표현할 수 있다. 클래스와 클래스간의 관계는 선으로 이어진다.

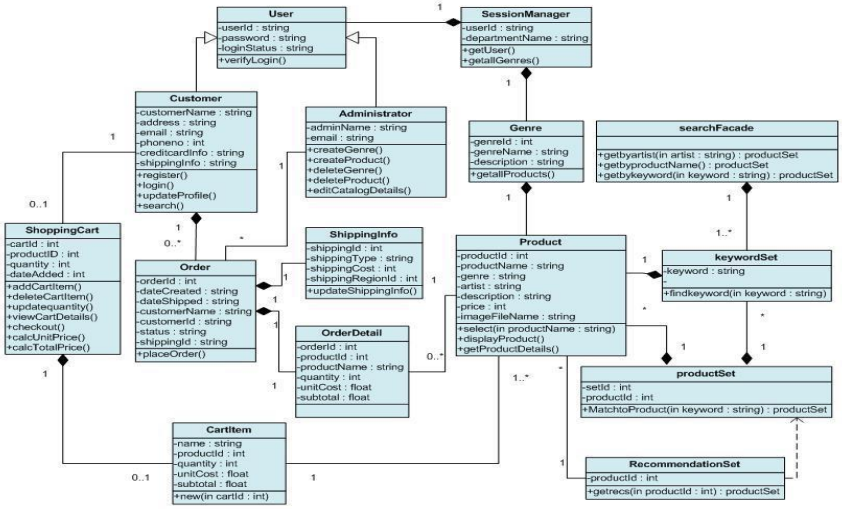

그림 1 Class diagram

2.2.3. Package Diagrams

복잡한 시스템을 이해하는 방법은 추상적인 개념들을 하나의 그룹으로 묶어서 이해하는 것이다. 이렇게 만든 그룹은 클래스와 같은 추상 개념으로 많은 인스턴스들을 가지고, 오로지 시스템을 이해하기 위한 목적으로만 존재한다. UML에서 시스템을 이해하기 위한 목적으로 추상적인 개념들을 모은 하나의 그룹을 패키지(Package)라고 한다. 패키지는 요소들의 그룹으로 조직하기 위한 범용 메커니즘으로 모델의 요소들을 조직하고 이해할 수 있도록 해준다. 패키지에 안에 담기는 것은 클래스에만 국한되는 것은 아니며, Use Case, Activity Diagram 등도 담을 수 있고, 다른 패키지도 담을 수 있다. 패키지를 구성할 때에는 여러 사람이 동의할 수 있는 형태로 구성되어야 하며, 패키지의 구성과 이름 체계는 개발자들이 쉽게 이해하고 사용할 수 있어야 한다.

패키지 내부의 모든 클래스들은 개념적, 기능적, 변화적, 관리적 측면에서 유사한 면을 가진다. 하나의 패키지 내부의 클래스들은 밀접한 관련성을 가지며, 다른 패키지의 클래스들과는 약한 의존관계가 있다. 이와 같이 패키지 다이어그램은 패키지 삽입 및 패키지 확장을 포함하여 모델 요소들을 그룹화(패키지화)함으로써 조직 및 요소의 독립성을 묘사한다. 또한 패키지 다이어그램은 요소들간의 관계를 시각화하여 제공한다.

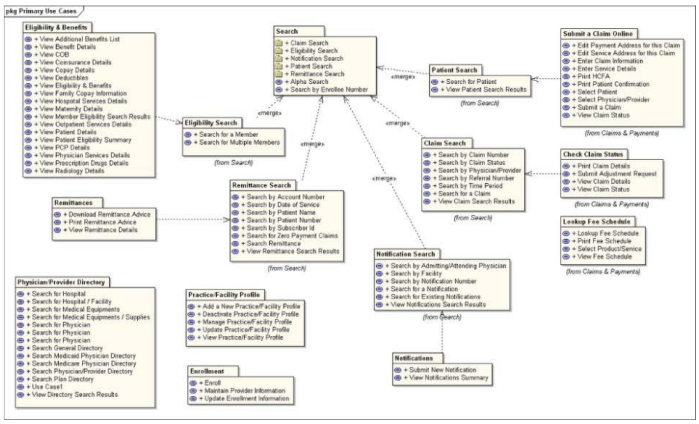

그림 2 Package diagram

2.2.4. Sequence Diagrams

시스템 아키텍쳐에 요구되는 운영 또는 결과에 영향을 주는 공동 작업 내에서 객체간의 메시지 집합의 교환 같은 상호작용(interaction)작업을 표현하기 위해서 사용되는 것이 순차 다이어그램이다. 순차 다이어그램에서, 다이어그램의 맨 위는 객체 또는 actor가 놓인다. 각 객체는 아래로 길게 늘어뜨린 Life Line을 가지고 있으며, 객체는 자신의 Life Line에 의하여 상호교환 작용에 관여하게 된다. 이름이 있는 화살표는 Message를 의미하며, 이 다이어그램에서 시간의 순서에 따른 상호작용을 나타내며 Message의 순서는 위에서 아래로, 수행된다. 수신 객체가 오퍼레이션을 완료할 때까지 송신 객체는 기다리고 있다.

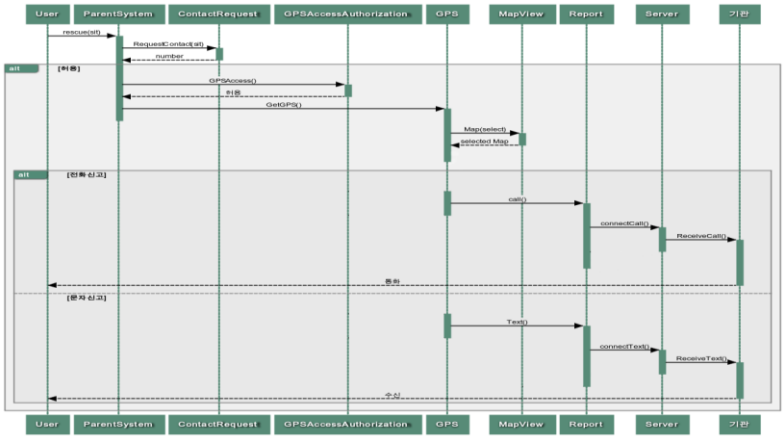

그림 3 Sequence diagram

2.2.5. State Diagrams

State Diagram은 객체지향 모델에서 클래스의 인스턴스 사건(Event)에 의거한 시스템의 전체적인 작동을 상세히 기술하는 UML의 다이어그램이다. State Diagram은 상태의 변화에 의한 동작 또는 하나의 상태에서 다른 상태로 변화되게 하는 사건의 주어진 시간 동안의 상태를 나타낸다. 이러한 표기는 실제 구현에서 UI를 정의하는 데 도움을 준다.

상태 다이어그램은 어떤 이벤트에 대한 반응적인(Reactive) 특성을 가지는 객체에 대해 기술하기 위해 이용되며, 객체가 갖는 여러 가지 상태를 표현한다. 객체는 기존 상태에 따라 동일한 메시지에 대해서 다른 행동을 보이기 때문에, 상태 다이어그램을 통해 시스템 내의 객체가 가질 수 있는 상태가 어떤 것이 있는지에 대해, 또 각 상태를 나타낼 때 특정 이벤트에 대해 어떤 반응을 보일지에 대해 기술한다.

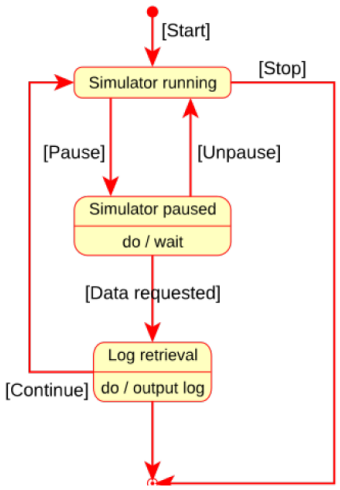

그림 4 State diagram

2.3. Applied Tool

2.3.1. GenMyModel

그림 5 GenMyModel.com

설계 명세서는 GenMyModel을 통해 Sequence Diagram을 제외한 모든 다이어그램을 설계하였다. 해당 도구는 온라인 상에서 사용 가능한 UML 설계 도구로써 class diagram, state diagram에 작성되는 변수, 메소드, 상태명들을 모두 확인하고 연결 가능하여 효과적인 설계가 가능하다. 또한 온라인에서 사용할 수 있기 때문에 어디에서든 작업 가능한 용이성을 띄고 있다.

2.3.2. Sequence Diagram.org

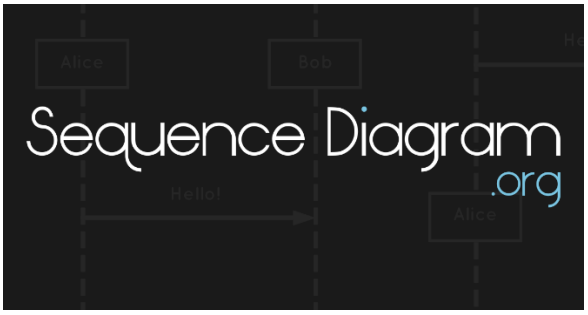

그림 6 SequenceDiagram.org

해당 명세서의 sequence diagram은 SequenceDiagram.org를 통해 설계하였다. 온라인에서 작업이 가능하며 sequence diagram을 작성한 이미지와 관련 xml 파일까지 추출할 수 있어 수정이 용이하다는 면에서 해당 도구를 사용하였다.

3. System architecture

3.1 Objective

System Architecture 에서는 우리 팀에서 개발하고자 하는 시스템에 대해 전반적으로 서술한다. 시스템의 전체적인 구조를 설명한다. 또, 시스템을 Block diagram 으로 나타내어, 각각의 관계와

실제 어떻게 사용되는 지를 Package diagram 을 사용하여 설명한다. 각 시스템에 대한 modular decomposition 은 4~9 장에 걸쳐 설명한다.

3.2 System Architecture

System Architecture에서는 애플리케이션 내의 서브 시스템들의 모델과 사용하는 데이터베이스간의 모델을 제공한다. ISES은 애플리케이션과 데이터베이스로 구성된다. 애플리케이션과 데이터베이스간의 연결은 I/O system을 통해 이루어진다.

3.2.1. 애플리케이션

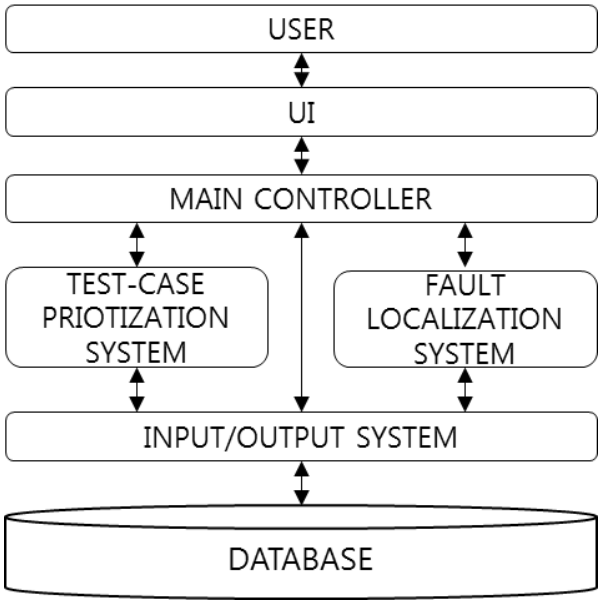

Diagram 1 ISES System architecture

A. user

ISES을 이용하는 사용자

B. UI

User Interface를 의미한다.

C. Test case prioritizer

ISES의 테스트케이스 우선순위화 기능을 담당하며, 과거 결함 추적 정보를 분석하는 기능과 우선순위화 수행 기능을 갖는다.

D. Fault Localizer

ISES의 결함 추적 기능을 담당하며, 테스트 케이스 수행 정보 분석 기능, 65가지의 알고리즘을 갖는 추적 기능을 갖는다

E. I/O

ISES의 파일과 테스트케이스를 데이터베이스로부터 받아오거나, 결함 추적 결과를 파일로 저장하기 위해 사용된다.

F. Database

ISES을 통해 사용되거나 산출된 정보들을 기록하는데 사용된다.

3.2.2. Test-case prioritization system

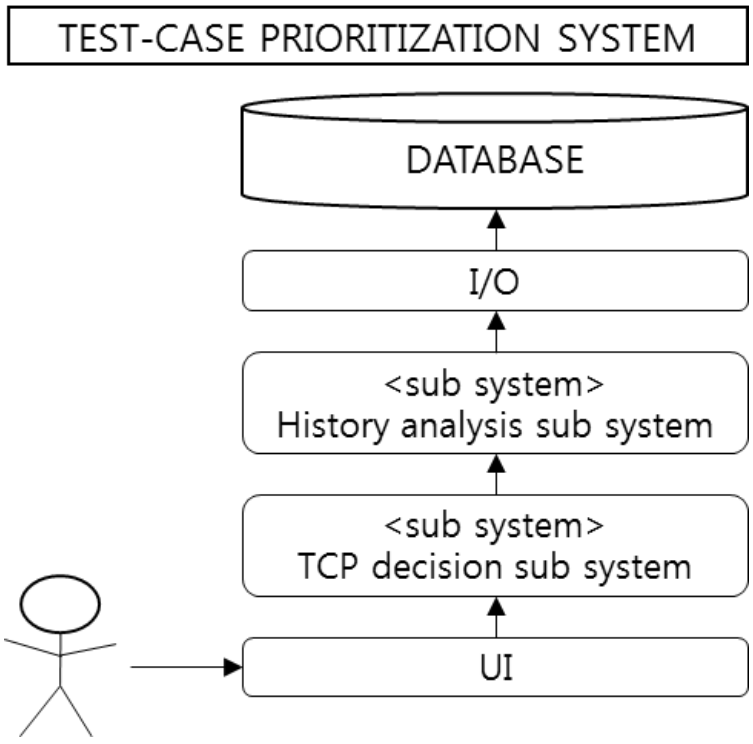

Diagram 2 TEST-CASE prioritizer system architecture

A. TCP decision sub system

테스트 케이스 우선순위화의 수행 여부 및 알고리즘을 결정하는 기능을 담당한다

B. History analysis sub system

과거 동일한 파일에 대해 결함을 추적했던 결과들을 분석한다

C. I/O

유저로부터 입력된 파일과 동일한 파일을 불러온다.

3.2.3. Fault localizer

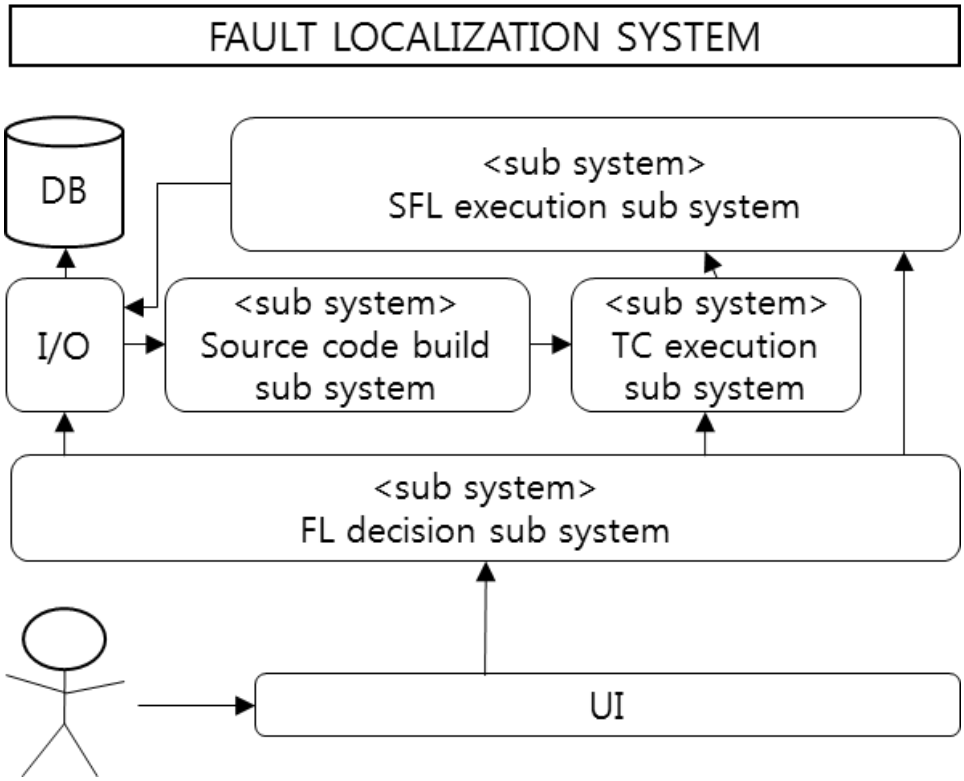

Diagram 3 Fault localizer system architecture

A. Source code build sub system

유저가 입력한 파일을 불러와 자동으로 빌드하고 수행가능한 프로그램으로 만드는 기능을 담당한다.

B. Test-case execution sub system

유저가 입력한 테스트케이스를 불러와 수행 가능한 프로그램의 입력값으로 수행시키는 기능을 담당한다.

C. Spectrum based fault localization execution sub system

테스트 케이스 수행 결과와 유저가 선택한 SFL 알고리즘을 통해 결함 위치를 추적하기 위한 계산을 수행하는 기능을 담당한다.

D. I/O

유저가 등록한 파일과 테스트 케이스를 가져오고 추적 결과를 데이터베이스에 저장하는 기능을 담당한다.

3.3 Package Diagram

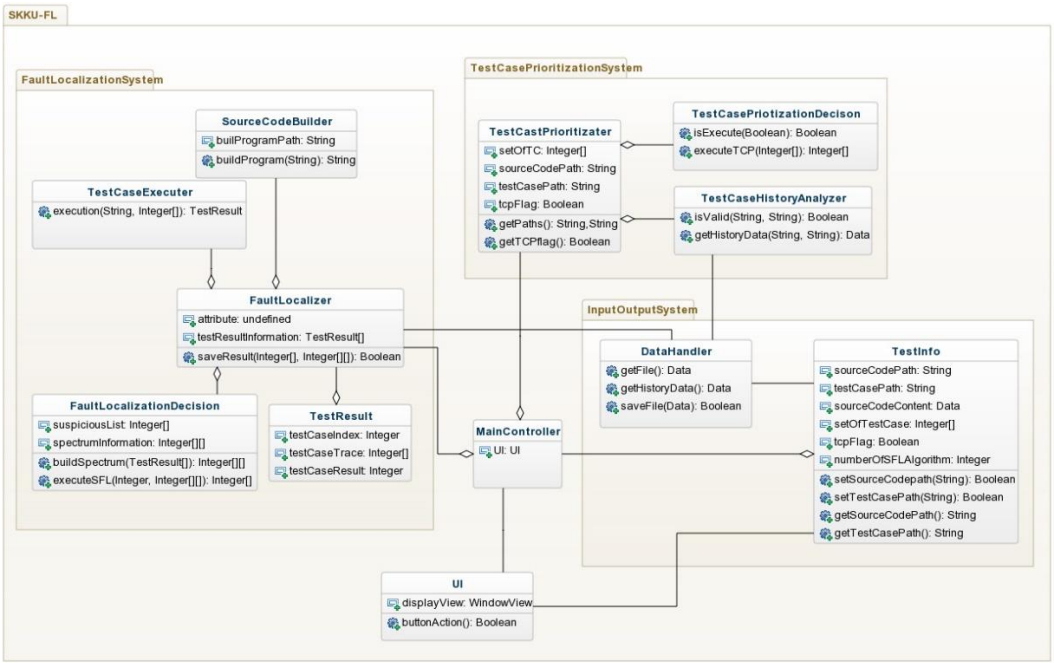

Diagram 4 ISES system package diagram

4. Test-case prioritization system

4.1. Objectives

유저가 디버깅 시간을 단축하기 위해 테스트케이스 우선순위화 기술을 사용 여부를 결정하면, 이에 따라 등록된 테스트 케이스를 우선순위화하는 기능을 하는 테스트 케이스 우선순위화 시스템의 설계를 설명한다. Class diagram, sequence diagram과 state Diagram을 통해 Test-case Prioritization System의 구조를 표현하고 설명한다.

4.2. Class diagram

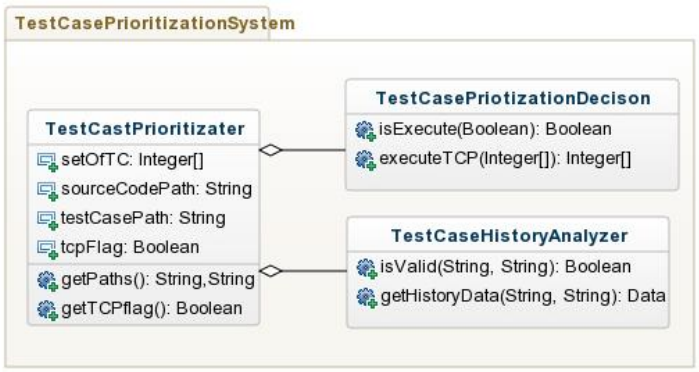

Diagram 5 Test Case Priotization system class diagram

4.2.1. Test Case prioritization decision

A. Methods

+ Boolean isExecute(Boolean tcpFlag): 사용자로부터 TCP 수행 여부에 대한 메시지를 받아 이를 확인하고 우선순위화를 수행한다..

+ Integer[] executeTCP(Integer[] previousSetofTCs): 무작위로 나열된 현재 테스트케이스들에 대해 과거 정보에 기반하여 테스트 케이스 우선순위화를 수행하고, 우선순위화 된 테스트케이스 집합을 반환 받는다.

4.2.2. Test Case History analyzer

A. Methods

+ Boolean isValid(String sourceCodePath, String testCasePath): 사용자로부터 입력받은 소스코드와 테스트 케이스의 파일명을 통해 해당 소스코드와 테스트 케이스에 대한 과거 정보가 있는지 확인하고 존재 여부를 반환한다.

+ Data getHistoryData(String sourceCodePath, String testCasePath): 사용자가 입력한 소스코드와 테스트 케이스의 파일명을 통해 해당 소스코드와 테스트 케이스에 대한 과거 정보를 반환한다.

4.2.3. Test Case Prioritizater

A. Attributes

+ Integer[] setOfTC: 테스트 케이스들의 집합

+ String sourceCodePath: 유저가 입력한 소스코드의 위치 및 파일명

+ String testCasePath: 유저가 입력한 테스트케이스의 위치 및 파일명

+ Boolean tcpFlag: 유저가 TCP 수행의 선택 여부

B. Methods

+ String[] getPaths(): 유저가 입력한 소스코드와 테스트 케이스의 위치 및 파일명을 가져온다

+ Boolean getTCPflag(): 유저가 선택한 TCP 수행 여부를 가져온다

4.3. Sequence Diagram

TCP 시스템내의 클래스 중 TestCasePrioritizationDecision 과 TestCaseHistoryAnalyzer 는 TestCasePrioritizater 가 상속받고 있으므로 모든 활동은 TestCasePrioritizater 로 의해 수행된다. 해당 클래스는 I/O system 을 통해 파일의 내용을 요청하여 받기 때문에 I/O sytem 과의 관계도 함께 명시한다.

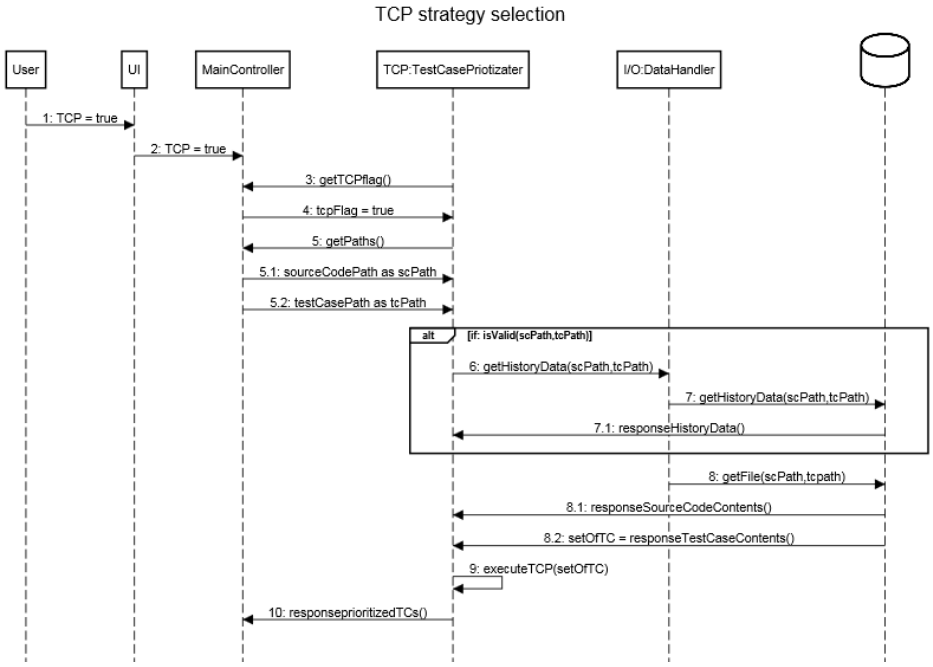

Diagram 6 TCP strategy selection sequence diagram

4.4. State Diagram

해당 시스템의 상태들은 유저가 TCP 수행을 원하는지 대기하는 상태(wait for TCPrequest), 과거 정보를 분석하기 위해 I/O 시스템으로 데이터를 받을 때까지 대기하는 상태(wait for historyData),

해당 정보가 우선순위화를 위해 사용될 수 있는지 확인하는 상태(check database), 과거 정보가 존재하지 않을 때 우선순위화를 적용할 수 없는 상태(error message), TCP 를 적용한 후 완료되는 상태 (apply TCP)로 구성된다.

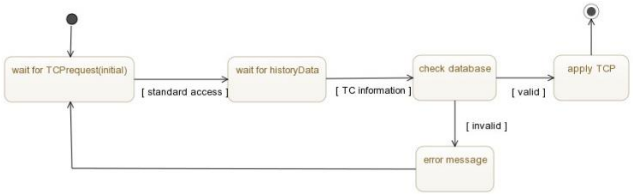

Diagram 7 test case prioritization state diagram

5. Fault localization system

5.1. Objectives

유저가 등록한 소스코드를 수행 가능한 시스템으로 빌드하고 빌드된 프로그램에 선택된 테스트케이스들을 수행시켜 얻은 수행 정보 및 결과를 토대로 스펙트럼을 생성하고, 생성된 스펙트럼에 대해 유저가 선택한 SFL 알고리즘을 통해 소스코드의 라인별 결함 의심도를 계산하는 결함 추적 시스템의 설계를 설명한다. Class diagram, sequence diagram과 state Diagram을 통해 Test-case priotizer System의 구조를 표현하고 설명한다.

5.2. Class diagram

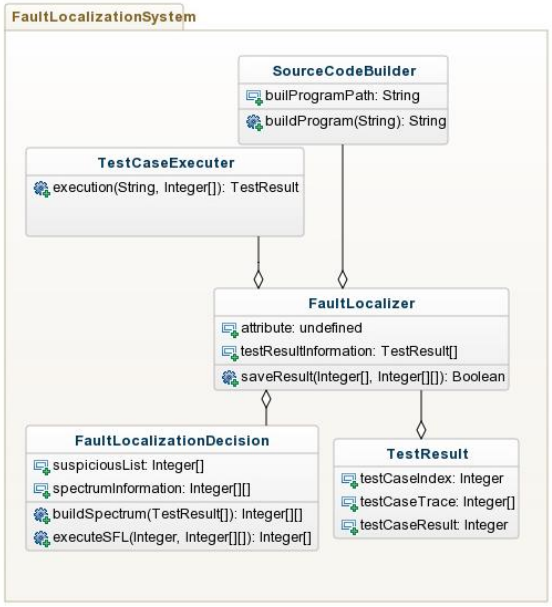

Diagram 8 Fault localizer sub system class diagram

5.2.1. Test result

A. Attributes

+ Integer testCaseIndex: 테스트케이스 집합들 내의 개별 테스트 케이스의 인덱스

+ Integer[] testCaseTrace: 개별 테스트 케이스를 수행시켰을 때 소스코드의 개별 라인을 지나갔는지 아닌지 여부의 정보 (트레이스 정보) (지나간 경우 1, 그렇지 않은 경우 0)

+ Integer testCaseResult: 개별 테스트 케이스를 수행시킨 후 테스트 케이스에 작성된 예상 결과와 실제 결과의 매칭을 통한 결과 (서로 같을 경우 1, 그렇지 않은 경우 0)

5.2.2. FaultLocalizer

A. Attributes

+ TestResult[] testResultInformation: 개별 테스트 케이스의 수행 정보와 결과를 갖는 TestResult 클래스들의 집합

B. Methods

+ Boolean saveResult(suspiciousList[], spectrumInformation): 선택된 소스코드와 테스트 케이스들을 수행시켜 계산하여 얻은 소스코드의 개별 라인 별 의심도와 생성된 스펙트럼 정보를 파일로 저장한다.

5.2.3. SourceCodeBuilder

A. Attributes

+ String buildProgrampath: 소스코드를 수행가능한 프로그램으로 빌드한 후 생성된 프로그램의 경로

B. Methods

+ buildProgram(sourceCodePath): 소스코드의 경로를 통해 해당 소스코드를 수행가능한 프로그램으로 빌드한다.

5.2.4. Test Case Executer

A. Methods

+TestResult execution(buildProgramPath, setOfTC): 빌드된 프로그램에 유저가 선택한 테스트 케이스들의 집합을 수행시켜 TestResult 클래스들을 생성한다.

5.2.5. Fault localization decision

A. Attributes

+ Integer[] suspiciousList: 소스코드의 라인 별 SFL 을 적용해 계산된 의심도 배열

+ Integer[] spectrumInformation: 소스코드에 개별 테스트케이스를 수행 후 얻은 정보들을 통합하여 생성된 스펙트럼 정보 (지나가고 pass/지나가고 fail/안지나가고 pass/안지나가고 fail)

B. Methods

+ Integer[] buildSpectrum(TestResult[]): 테스트 케이스 수행 내역들과 결과를 통해 스펙트럼 정보를 생성

+ Integer[] executeSFL(sflAlgorithmIndex, spectrumIntormation[]): 유저가 선택한 알고리즘과 생성된 스펙트럼 정보를 통해 SFL 을 수행하여 suspiciouslist 를 생성

5.3. Sequence Diagram

SFL 시스템내의 클래스 중 SourceCodeBuilder, TestCaseExecutor, FaultLocalizationDecision 은 FaultLocalizer 가 상속받고 있으므로 모든 활동은 FaultLocalizer 로 의해 수행된다. 해당 클래스는 I/O system 의 TestInfo 를 통해 유저가 선택한 정보를 얻고 dataHandler 를 통해 파일을 저장하기 때문에 I/O system 과의 관계도 함께 명시한다. 유저가 UI 에 등록한 후 관리되는 정보들에 대해서는 I/O system 설계에서 구체적으로 설명한다.

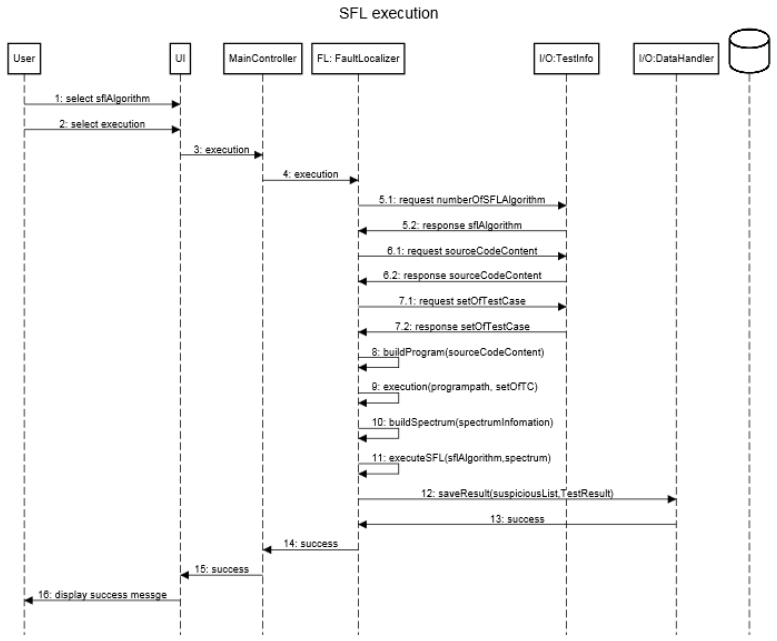

Diagram 9 SFL Execution sequence diagram

5.4. State Diagram

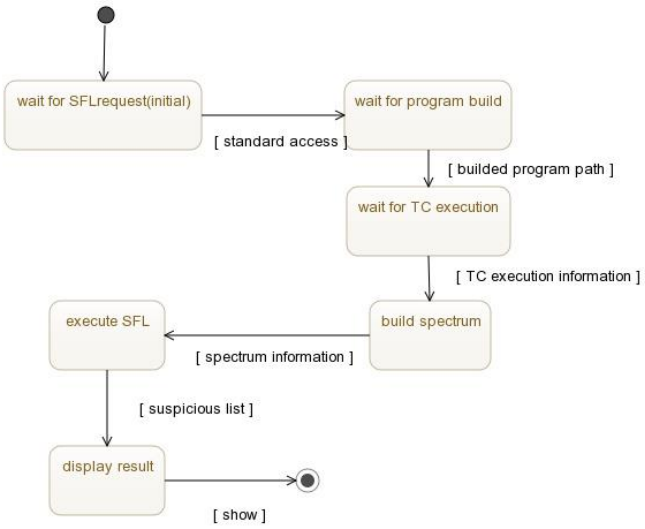

Diagram 10 Fault Localator state diagram

6. Input/Output system

6.1. Objectives

ISES 을 위해 유저가 등록한 소스코드와 테스트 케이스의 정보들을 관리하고 결함 추적 결과를 저장하기 위한 입출력 시스템에 대해 설계한다. Class diagram, sequence diagram 과 state Diagram 을 통해 Test-case priotizer System 의 구조를 표현하고 설명한다

6.2. Class diagram

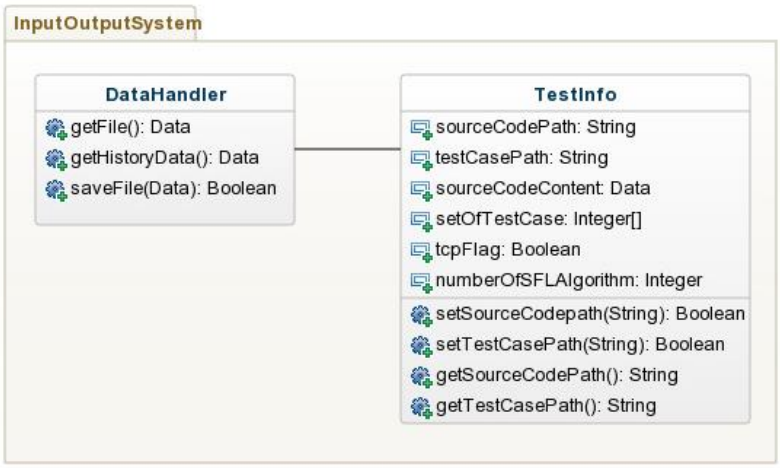

Diagram 11 Input/Output System class diagram

6.2.1. DataHandler

A. Attributes

해당 사항 없음

B. Methods

+ Data getFile(path): 입력되는 경로에 있는 파일의 내용을 가져온다

+ Data getHistoryData(sourceCodePath, testCasePath): 입력된 소스코드와 테스트 케이스들의 정보를 토대로 과거 분석 결과를 가져온다.

+ Boolean saveFile(Data): 입력되는 데이터를 파일로 저장한다..

6.2.2. TestInfo

A. Attributes

+ String sourceCodePath: 유저가 등록한 소스코드의 경로

+ String testCasePath: 유저가 등록한 테스트 케이스의 경로

+ Data sourceCodeContent: 유저가 등록한 소스코드에 담긴 내용들

+ Integer[] setOfTestCases: 유저가 등록한 테스트 케이스들의 인덱스 집합

+ Boolean tcpFlag: 유저가 선택한 TCP 수행 여부

+ Integer numberOfSFLAlgorithm: 유저가 선택한 SFL 알고리즘의 인덱스

B. Methods

+ Boolean setSourceCodePath(sourceCodePath): 유저가 등록한 소스코드의 경로를 저장한다

+ Boolean setTestCasePath(testCasePath): 유저가 등록한 테스트 케이스의 경로를 저장한다.

+ String getSourceCodePath(): UI 로부터 유저가 등록한 소스코드의 경로를 가져온다.

+ String getTestCasePath(): UI 로부터 유저가 등록한 테스트 케이스의 경로를 가져온다.

6.3. Sequence Diagram

I/O 시스템은 유저가 소스코드와 테스트 케이스의 경로 및 SFL 알고리즘, TCP 여부를 입력하면 관련 정보를 담기 위한 TestInfo 클래스를 생성하고, SFL 결과를 저장하는 기능을 수행한다. SFL

결과를 저장하는 기능은 5 장에서 설명하였다. 해당 다이어그램은 Test Info 클래스를 생성하는 과정을 설명한다.

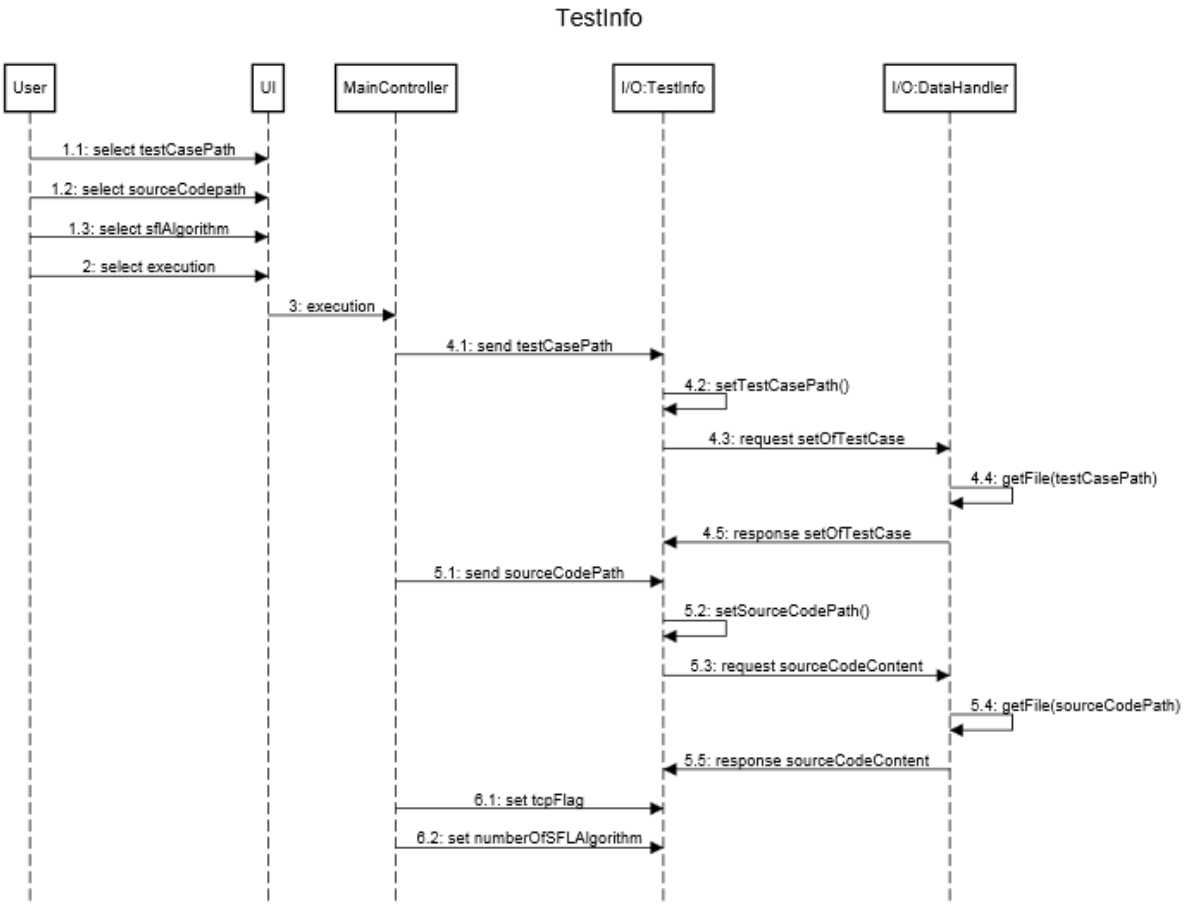

Diagram 12 Input/Output System sequence diagram

7. User Interface Design

7.1. Objectives

User interface design 은 GUI 로 제공될 ISES 시스템에 대한 화면에 대해 설명한다. 해당 설명을 위해 ISES 의 화면을 설계하고 각 세부 화면에 대해 설명한다.

7.2. Graphic User Interface

향후 개발하기 위해 남겨둔 기능 외에 현재 버전에서 개발할 화면에 대해서만 설명한다. 크게 유저가 결함 추적을 수행하기 이전에 나타나는 화면인 Test Case Selection 과 수행 이후에 유저에게 제공될 화면인 Test Result 로 나눈다.

7.2.1. Test Case Selection

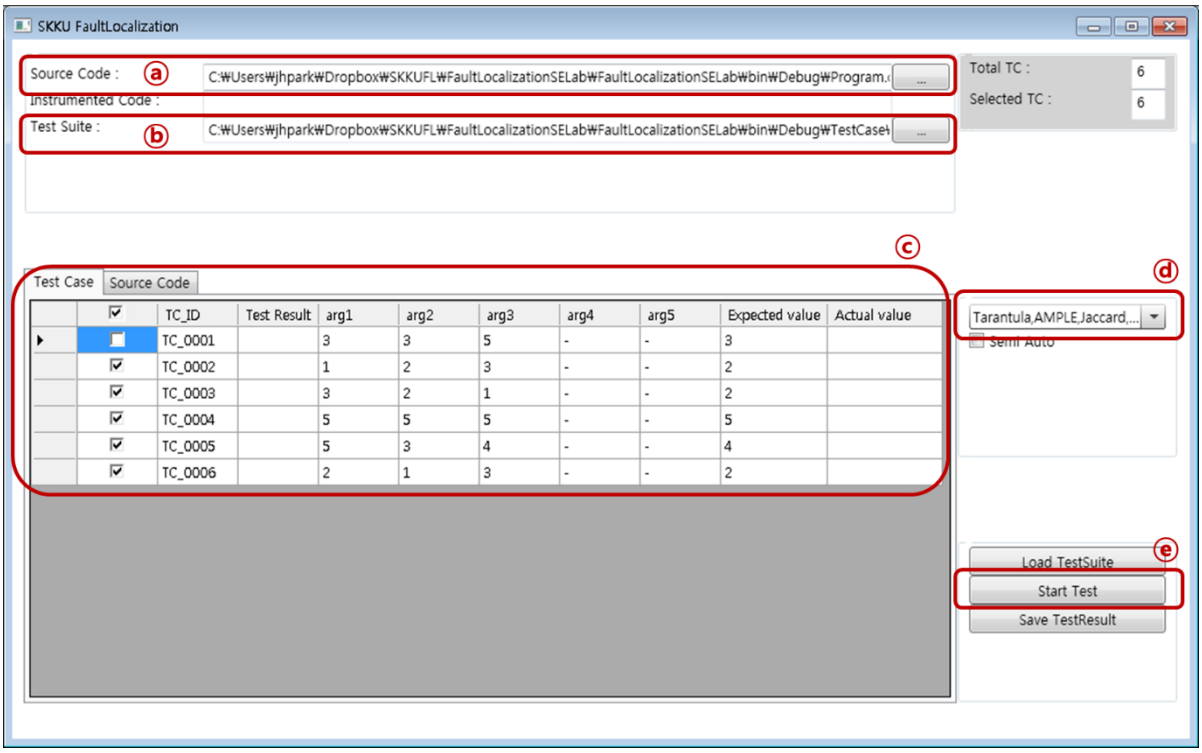

그림 7 ISES UI example 1

A. 소스코드 경로 및 파일명 선택

유저는 해당 항목에 대해 "..." 아이콘을 클릭하여 결함을 추적하고자 하는 소스코드의 경로와 파일명을 선택한다.

B. 테스트 케이스의 경로 및 파일 선택

유저는 해당 항목에 대해 "..." 아이콘을 클릭하여 결함 추적에 사용하기 위한 테스트 케이스의 경로와 파일명을 선택한다. 선택된 이후에 ©와 같은 화면이 활성화 된다.

C. 테스트 케이스들의 선택

유저가 등록한 테스트 케이스의 경로에 존재하는 모든 테스트 케이스들을 불러오고 해당 내용들을 확인한다. 좌측 체크박스를 통해 테스트 케이스들의 선택 여부를 결정할 수 있다.

D. SFL 알고리즘의 선택

결함 추적에 사용하기 위한 SFL 알고리즘을 선택한다.

(E. TCP 수행 여부의 선택

결함 추적 시간을 줄이기 위해 TCP 를 수행할 지 하지 않을지 여부를 선택한다.)

E. 결함 추적 실행

소스코드, 테스트 케이스의 선택이 완료된 후 해당 아이콘을 클릭하면 유저가 선택한 소스코드를 자동으로 빌드하고 빌드된 프로그램에 선택한 테스트 케이스를 자동으로 수행시키고, 수행 결과에 선택된 SFL 알고리즘을 적용하여 소스코드의 라인별 결함 의심도를 계산한다.

7.2.2. Test Results

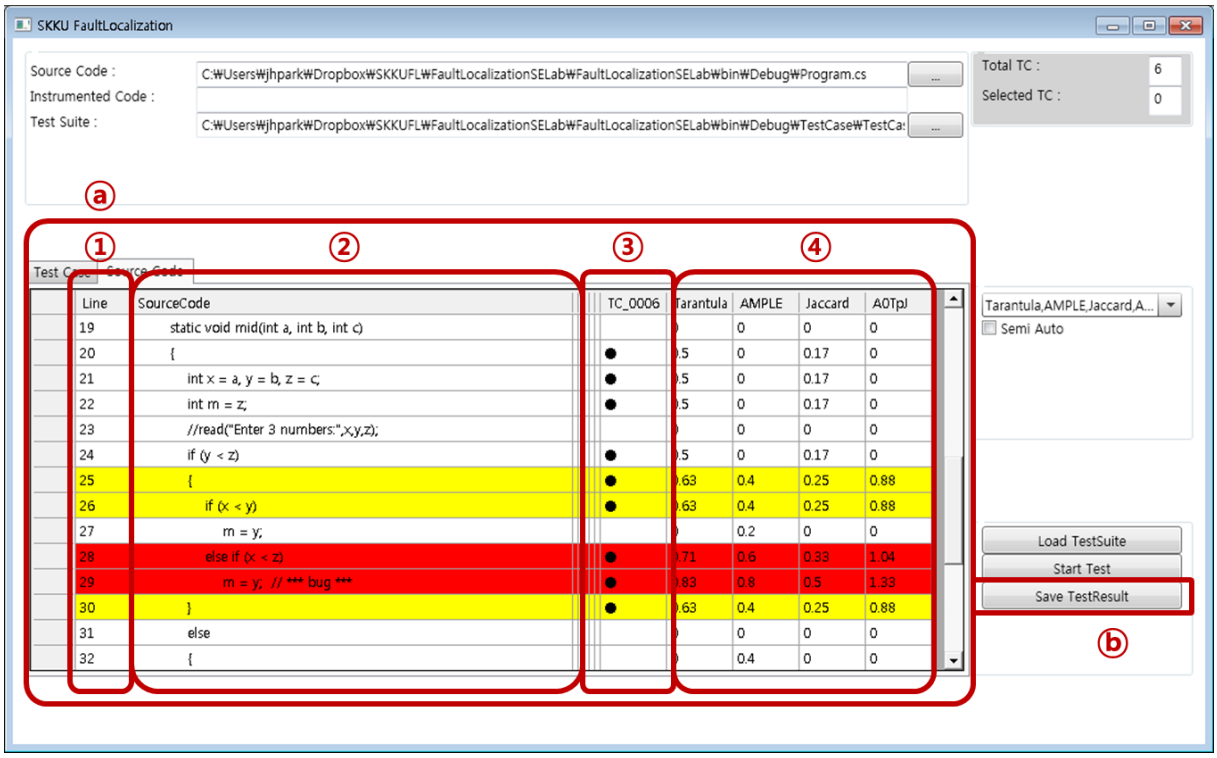

그림 8 ISES UI example 2

A. 소스코드의 라인별 의심도 시각화

결함 추적을 수행한 후에 산출된 결과를 유저에게 시각적으로 제공하기 위한 창이다. 해당 창은 소스코드의 라인 번호, 소스코드 내용, 적용된 테스트 케이스가 지나간 라인의 정보, 유저가 선택한 SFL 알고리즘을 통해 계산된 각 라인별 의심도로 구성되며, 계산된 의심도의 순위에 따라 의심되는 라인을 눈에 띄게 표현한다.

A.1. 소스코드 라인 정보

유저가 선택한 소스코드 파일 내용에 개별 라인 번호를 보여준다.

A.2. 소스코드 정보

유저가 선택한 소스코드의 내용을 보여준다.

A.3. 테스트 케이스 수행 정보

유저가 선택한 테스트 케이스가 해당 소스코드의 몇번째 라인들을 지나갔는지의 여부를 "●"로 표시한다.

A.4. SFL 알고리즘 수행 결과

유저가 선택한 SFL 알고리즘을 수행하여 계산된 개별 라인의 결함 의심도를 표시한다. 해당 의심도들은 전체적으로 순위화 되어 결함으로 더욱 의심되는 라인을 돋보이도록 돕는다.

B. 테스트 결과 저장

A 에 표시되는 모든 내용들을 엑셀파일로 저장을 수행한다.

8. Testing Plan

8.1. Objectives

시스템이 의도한 방향으로 실행되고 시스템 내부의 결함을 찾기 위해 testing 을 한다. 이를 위해 설계단계에 미리 계획한다. 이 때 Testing Plan 에서는 Testing Policy 와 여러 Test Case 에 대해 기술한다.

8.2. Testing Policy

ISES 시스템의 개발에서는 크게 3 단계로 나누어 testing 을 진행한다. Developing Testing, Release Testing, User Testing 으로 나눠지며, Developing Testing 은 다시 Component Testing, Integrating Testing, System Testing, Acceptance Testing 의 4 단계로 나눠진다.

8.2.1. Developing Testing

개발 과정에서 수행되는 testing 으로 Component Testing 은 각 component(unit) 단위로 개발이 진행되면, 각 요소들이 개발한 후에 제대로 작동하는지 확인하는 testing 이다. Integrating Testing 은 Component Testing 후 각 요소들을 하나씩 점진적으로 합치면서 하는 testing 이다. System Testing 은 모든 하위 시스템을 하나로 합친 후, 그 시스템이 잘 동작하는지 testing 하는 것이다. Acceptance Testing 은 사용자의 정보를 이용하여 시스템에 대한 사용자의 요구사항을 testing 하는 것이다.

8.2.2. Release Testing

사용자에게 출시하기 전에 최종 시스템을 testing 하는 것이다. 요구사항 명세서에서 작성되었던 요구사항이 제대로 반영되었는지를 확인한다.

8.2.3. User Testing

사용자가 사용자의 환경에서 시스템을 testing 하는 것이다.

8.3. Test Case

8.3.1. Test-case prioritizater

A. Click “Start Test” without TCP check

A.1. User: 결함 추적을 위한 소스코드와 테스트 케이스를 선택하고 “TCP”를 체크하지 않고 “Start Test”를 클릭한다.

A.2. 시스템 동작: TCP 시스템은 동작하지 않으며 다른 시스템들이 수행된다.

B. Click “Start Test” with TCP check

B.1. User: 결함 추적을 위한 소스코드와 테스트 케이스를 선택하고 “TCP”를 체크하고 “Start Test”를 클릭한다.

B.2. 시스템 동작: 유저가 등록한 소스코드와 테스트 케이스 과거 정보를 통해 테스트 케이스를 우선순위화 한다.

B.2.1) 과거 정보가 있을 때: 과거 정보를 토대로 TCP 알고리즘을 적용하여 등록된 테스트 케이스들을 우선순위화 한다.

B.2.2) 과거 정보가 없을 때: 유저가 등록한 테스트 케이스가 나열된 기존 순서에 따라 정렬한다.

8.3.2. Fault localizer

A. Click “Start Test” with SFL Algorithm

A.1. User: 결함 추적을 위한 소스코드와 테스트 케이스를 선택하고 결함 추적에 사용할 SFL 알고리즘을 선택하고 “Start Test”를 클릭한다.

A.2. 시스템 동작: 소스코드를 자동으로 빌드하고 선택된 테스트 케이스들을 수행시킨 결과들을 스펙트럼으로 생성하고, 유저가 선택한 SFL 알고리즘을 수행하여 소스코드의 라인별 의심도를 계산하고 시각화 한다.

B. Click “Save Result” after “Start Test”

B.1. User: 시각화된 정보가 유저에게 제공된 이후 이를 파일로 저장하기 위해 "Save Result"를 클릭한다.

B.2. 시스템 동작: 유저가 저장하고자 하는 파일 위치를 입력받고. Input/Output System 에게 해당 정보들을 저장하기를 요청한다.

9. Development Environment

9.1. Objectives

Development Environment 에서는 개발자의 환경에 대해 설명한다. 사용한 프로그래밍 언어와 IDE 에 대해 서술한다.

9.2. Programming language & IDE

9.2.1. Programming language – C#

C#은 마이크로 소프트웨어에서 개발된 언어로, 모든 것을 객체로 취급하는 커몦넌트 프로그래밍 언어이다. 기본적인 개발 언어인 C 를 확장한 C++에 기본을 두고, 닷넷(.net)플랫폼을 위해 개발되었다. 해당 언어는 C++에 기반을 두면서도, 비주억 베이직이나 자바와도 비슷하기 때문에, 이들 모두의 장점을 지닌다. 즉 비주얼 언어가 가진 사용자 친화성, C++의 객체 지향성, 자바의 분산환경 처리에 적합한 다중성등을 모두 지니는 컴포넌트 기반의 소프트웨어 개발 패러다임을 반영한다. 이 언어를 통해 하나 이상의 OS 에서도 사용할 수 있는 응용 프로그램을 만들 수 있기 때문에, 소프트웨어나 서비스를 값싸고 빠르게 시장에 내놓을 수 있다.

9.2.2. IDE – Visual Studio 2015

C#에 기반하여 ISES 을 구현하기 위한 IDE 로 Visual Studio 2015 를 사용하였다. Visual Studio 는 microsoft 사에서 무료로 제공하고 있는 개발 지원 도구로써, C#과 관련하여 효과적인 디버깅 기술을 제공해주고 있다.

9.3. Coding Rule

ISES 개발에 있어서 몇 가지 기본적인 지침을 제시하면 다음과 같다

- 코드를 명료하고 읽기 쉽게 작성한다.
- 코드를 일관성 있게 작성한다.
- 변수나 메소드 이름에 명료한 이름을 사용한다.
- 파일과 클래스들을 논리적으로 구성한다.

- 하나의 파일에 하나의 클래스만 작성한다(inner class 제외)
- 한 줄에 80여자이상은 쓰지 않는다.
- 공백문자들과 분리기호를 적절하게 잘 사용한다.

9.4. Version Management Tool

효율적인 개발과 코드 관리, 공유를 위해 GitHub 를 이용한다. GitHub 는 전세계적으로 많이 사용되는 웹 기반의 호스팅 서비스로 신뢰도가 높은 서비스이다. 또한 오픈 소스가 많이 등록되어 있어, 이를 이용하는 데 편리하고 유용한 서비스를 제공한다.

10. Introduction

10.1. 그림 Index

10.2. Diagram Index