Testing Report

T3

Marvin Habermann

오희수

정훈섭

진경훈

Team 6

목차

Our CTIP Environment

CTIP

JF JFeature

Deploy Server
Tomcat

CI & Build
Hudson & Ant

Unit Test & RE Tool
Junit & JFeature

SCM Tool
subclipse

eclipse

System Testing

System Testing

Testing the behavior of the whole software/system as defined in **software requirements specification(SRS)** is known as system testing, its main focus is to verify that **the customer requirements are fulfilled**.
System testing is done after integration testing is complete. System testing should test **functional and non functional requirements** of the software.

System Testing

Functional Requirements

- 파일실행, 파일삭제, 의존성 검사, 파일 수정, 파일 검색, 파일 정보 보기, Tag 생성, Tag 삭제, Tag 수정, Tag 지정, Tag 해제, Tag 의존성 부여, Tag 의존성 삭제, Tag 구조 보기, Service Directory, Name Tag 생성, Name Tag 삭제, Name Tag 수정, Name Tag 목록 보기, Log File 생성, Log File 보기

▶ Smoke Testing

Non-Functional Requirements

- User Interface를 직관적으로 구성한다
- 컴퓨터에서 기본적인 기능을 수행하기 때문에 dependability, reliability를 고려하여 만든다
- 단, TDMS 상에서 파일이 수정된 경우가 아니라면 dependability를 보장 할 수 없다

▶ Usability Testing

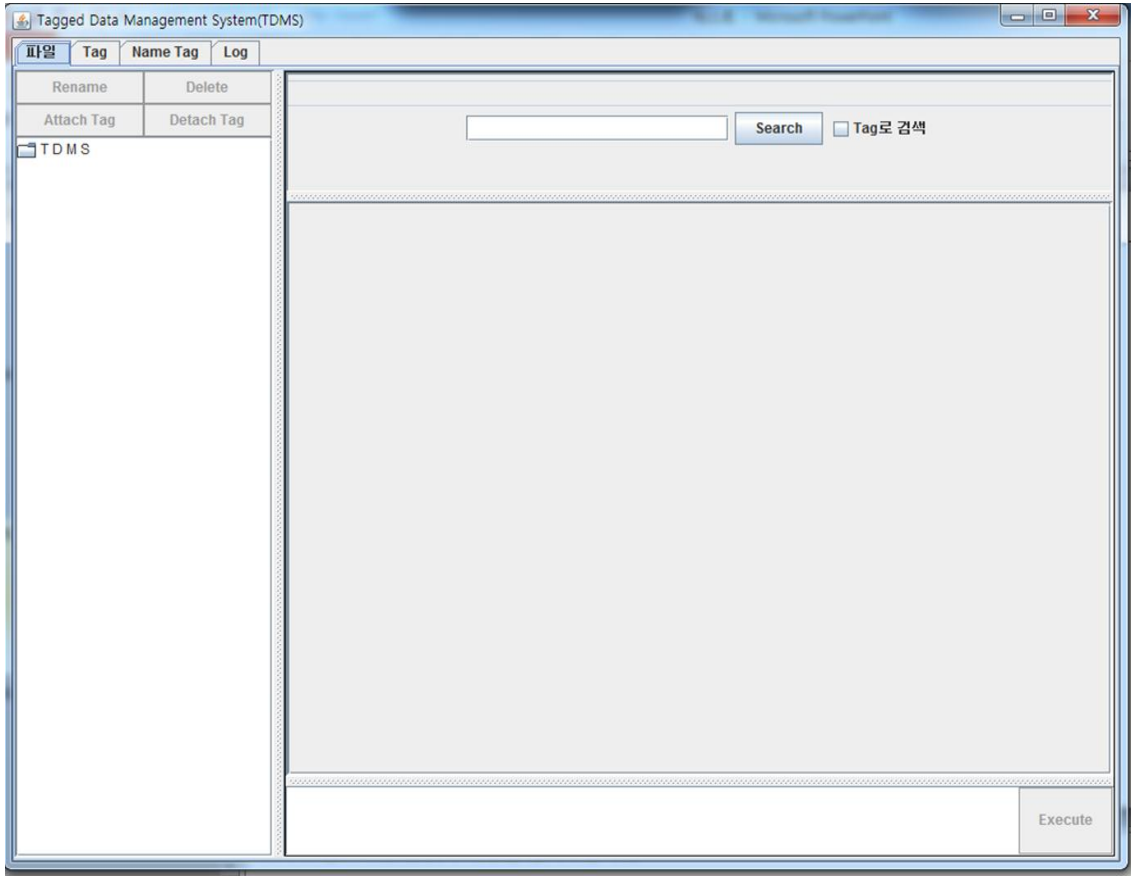
Smoke Testing

Smoke Testing is performed after software build to **ascertain that the critical functionalities of the program is working fine**. It is executed **"before"** any detailed functional or regression tests are executed on the software build. The **purpose is to reject a badly broken application**, so that the QA team does not waste time installing and testing the software application.

In Smoke Testing, the **test cases chosen cover the most important functionality** or component of the system. The objective is not to perform exhaustive testing, but to verify that the critical functionalities of the system is working fine. For Example a typical smoke test would be – Verify that the application launches successfully, Check that the GUI is responsive ... etc.

Smoke Testing

– 프로그램이 실행됩니까? PASS



IDE
: Eclipse 3.7.0

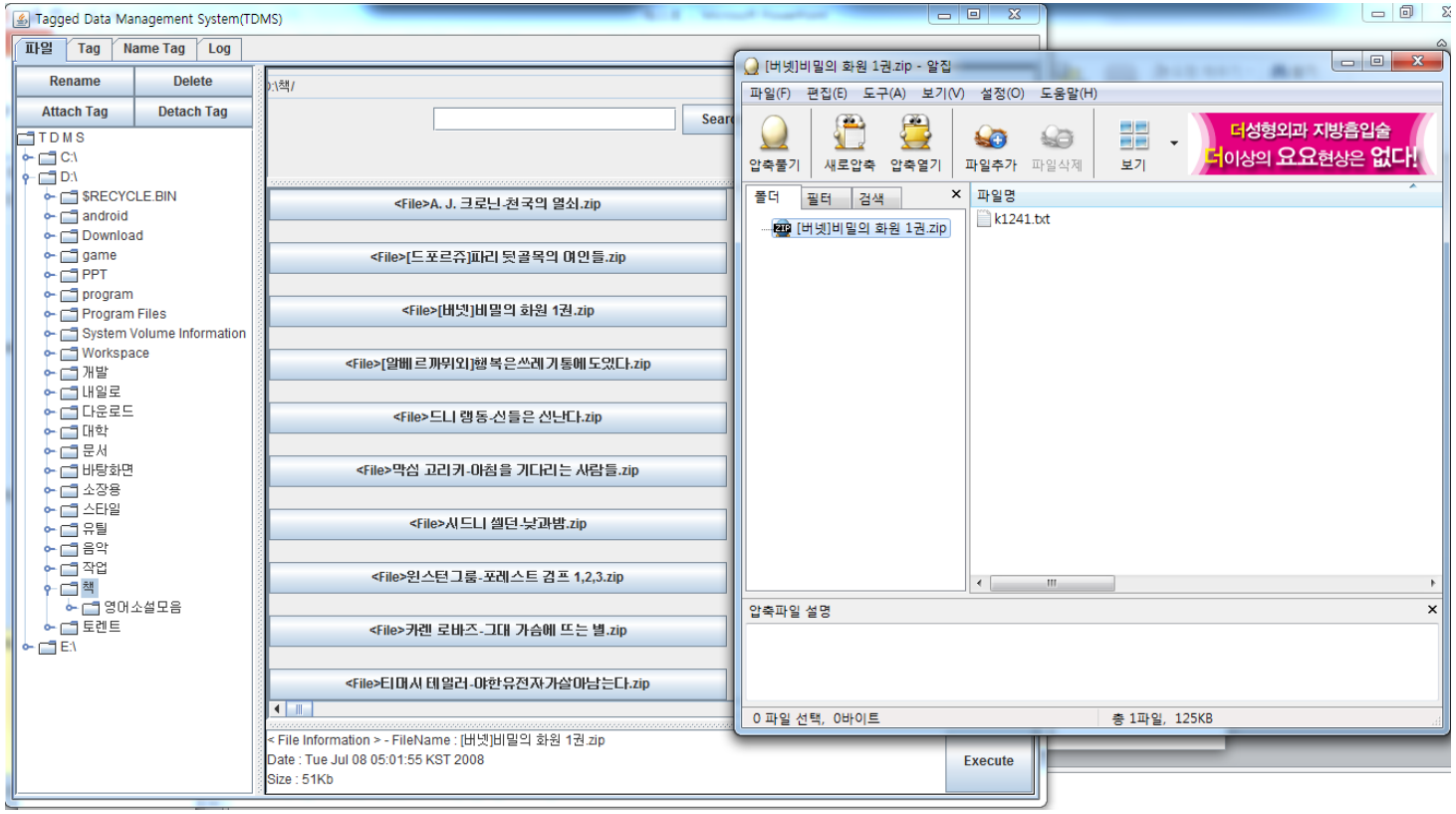
Compiler
: JAVA 1.7

Smoke Testing

Task	Testing
프로그램이 실행됩니까?	PASS
프로그램이 정상적으로 종료됩니까?	PASS
클릭되지 않는 버튼이 있습니까?	PASS
윈도우 창이 이동이 됩니까?	PASS
윈도우 창의 크기가 조절됩니까?	PASS
글씨가 깨지지 않고 제대로 출력됩니까?	PASS
실행 도중에 멈추지 않습니까?	PASS

Smoke Testing

– '파일실행' 기능이 실행됩니까? PASS



파일 탭 클릭 -> 폴더선택 -> 파일 선택-> Execute 버튼 실행

Smoke Testing

Task	Testing	Task	Testing
파일실행	PASS	Tag 해제	PASS
파일삭제	PASS	Tag 의존성 부여	FAIL
파일수정	FAIL	Tag 의존성 삭제	PASS
파일검색	PASS	Tag 구조 보기	FAIL
파일정보보기	PASS	Name Tag 생성	PASS
Tag 생성	PASS	Name Tag 삭제	PASS
Tag 삭제	FAIL	Name Tag 수정	PASS
Tag 수정	FAIL	Name Tag 목록보기	PASS
Tag 지정	PASS	Log File 보기	PASS

Smoke Testing

- '파일 수정' 기능이 실행됩니까? FAIL

1. 파일을 선택한다.
2. Rename버튼을 누른다.
3. 입력 창이 나오면 취소나 종료버튼을 누른다.
4. 선택된 파일의 이름이 null로 자동적으로 변경된다.

- 'TAG 삭제' 기능이 실행됩니까? FAIL

1. A와 B 라는 이름의 태그를 만든다.
2. B태그를 선택하고 Modify를 사용하여 이름을 A로 변경한다.
3. 뒤에 만들어진 A을 선택하고 Delete 버튼을 누른다.
4. 선택한 태그가 없어지는 게 아니라 먼저 만들어진 같은 이름의 A인 태그가 삭제된다.

Smoke Testing

– ‘TAG 구조보기’ 기능이 실행됩니까? FAIL

1. A, B, C 라는 이름을 가진 태그를 3개 만든다.
2. 태그 B를 태그 A와 의존성 부여를 한다.
3. 태그 C를 태그 B와 의존성 부여를 한다.
4. 연결된 태그 A와 B의 의존성을 삭제한다.
5. 의존성을 나타내는 선이 잘못된 방향을 가리킨다.

– ‘TAG 의존성 부여’ 기능이 실행됩니까? FAIL

1. A와 B 라는 이름의 태그를 만든다.
2. B태그를 선택하고 Modify를 사용하여 이름을 A로 변경한다.
3. 변경된 태그를 선택하여 의존성 부여버튼을 누른다.
4. 먼저 생성된 같은 이름의 다른 태그인 A을 선택한다.
5. 의존성이 만들어 지지 않는다.

Smoke Testing

- ‘TAG 수정’ 기능이 실행됩니까? FAIL

1. A와 B 라는 이름의 태그를 만든다.
2. B태그를 선택하고 Modify를 사용하여 이름을 A로 변경한다.
3. 먼저 만들어진 A의 속성이 B의 속성으로 복사된다.

- Coverage Report

Task	Count	Pass	Fail	Percent(%)
Common	7	7	0	100
Function	18	13	5	72
계	25	20	5	80

Usability Testing

Usability means the software's **capability to be learned** and **understood easily** and **how attractive** it looks to the end user.
Usability Testing is a **black box testing** technique.
Usability Testing tests the following features of the software.

1. How easy it is to use the software.
2. How easy it is to learn the software.
3. How convenient is the software to end user.

Usability Testing

Task 수행 시간 측정

Task	수행시간(s)	Task	수행시간(s)
파일실행	8	Tag 해제	10
파일삭제	9	Tag 의존성부여	6
파일수정	14	Tag 의존성 삭제	6
파일검색	170	Tag 구조 보기	1
파일정보보기	6	Name Tag 생성	15
Tag 생성	10	Name Tag 삭제	5
Tag 삭제	4	Name Tag 수정	8
Tag 수정	7	Name Tag 목록보기	1
Tag 지정	10	Log File 보기	3

Usability Testing

Interface 만족도 (파일 탭)

구성요소	만족도(10만점)	개선해야 할 부분
파일실행	7	파일과 폴더가 잘 구분되지 않는다.
파일삭제	7	파일과 폴더가 잘 구분되지 않는다.
파일수정	5	오류 시 안내가 없다
파일검색	2	대소문자 구분이 된다. 느리다
파일정보보기	7	파일과 폴더가 잘 구분되지 않는다.
Tag 지정	7	파일과 폴더가 잘 구분되지 않는다.
Tag 해제	5	Tag된 파일과 일반 파일의 구분이 안된다

Usability Testing

Interface 만족도 (Tag 탭)

구성요소	만족도(10만점)	개선해야할 부분
Tag 생성	8	특별한 점은 없다
Tag 삭제	8	특별한 점은 없다
Tag 수정	8	특별한 점은 없다
Tag 의존성부여	5	메뉴가 직관적이지 못하다
Tag 의존성 삭제	5	메뉴가 직관적이지 못하다
Tag 구조 보기	5	이상한 모양을 계속 만든다

Usability Testing

Interface 만족도 (Name Tag 탭)

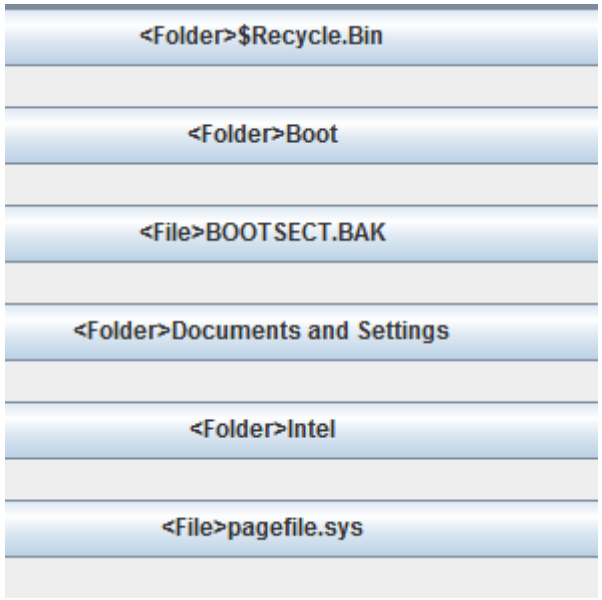
구성요소	만족도(10만점)	개선해야 할 부분
Name Tag 생성	5	파일,폴더선택이 어렵다
Name Tag 삭제	8	특별한 점은 없다
Name Tag 수정	5	파일,폴더선택이 어렵다
Name Tag 목록보기	10	직관적이지 못하다

Interface 만족도 (Log 탭)

구성요소	만족도(10만점)	개선해야 할 부분
Log File 보기	7	작업과 내용이 구분되면 좋겠다

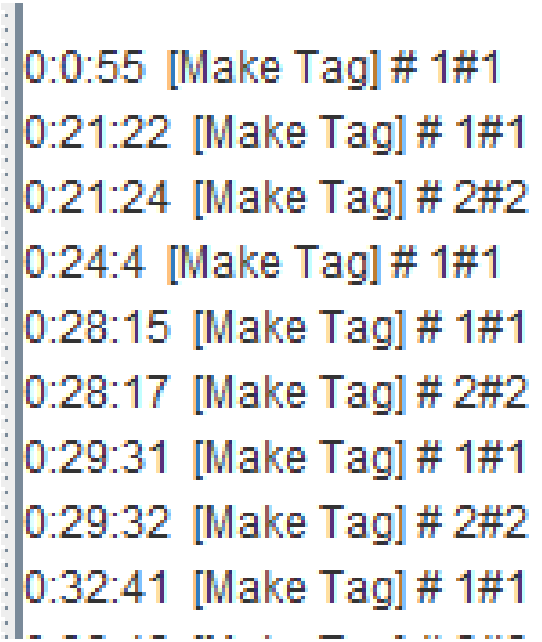
Usability Testing

오류, 원인 진단 & 개선 보완사항



파일과 폴더가 구분이 되지 않는다. 색을 사용하여 파일과 폴더를 각각 다른 색으로 표시하면 보다 직관적으로 구분할 수 있을 것 같다.

로그파일도 마찬가지로 수행한 작업과 내용이 구분될 수 있도록 색을 사용하던지, 기록된 내용이 어떤 작업인지 더 자세히 볼 수 있으면 좋겠다



Functional Specification

Identify Use Cases

먼저 시스템이 하는 기능을 큰 구조로 나누기 위하여
OSP stage 1000의 문서에서 Use Case를 참고한다.
이를 토대로 시스템이 수행하는 기능을 확인한다.

각각의 Use Case는 우리가 시스템을 테스트할 때
실제 테스트 대상이 되는 functional feature들을 뽑아
내기 위한 functional specification과 상응하게 된다.

Functional Specification

시스템에서는 크게 4가지의 영역에 대한 기능을 수행하게 되고, 그 기능은 다음과 같다.

파일

파일 실행	파일 삭제	파일 수정	파일 검색	파일정보보기

태그

태그 생성	태그 삭제	태그 수정	태그 검색	태그 지정
태그 해제	의존성 부여	의존성 삭제	태그구조보기	최근태그보기

네임태그

네임태그생성	네임태그삭제	네임태그수정	목록보기	로그파일보기

기타

로그파일보기

Retrieve Testable Features

Retrieve Testable Features

Functional Specification에서도 우리가 테스트 해야 할 기능들이 비교적 작게 나뉘어져 있지만, 실제로 테스트를 하기에는 각각의 Functional Specification들 역시 각각이 여러 기능을 일련적으로 수행하는 경우가 많다.

Functional Specification이 수행하는 일련의 활동을 다시 쪼개어 Testable Feature을 만든다. 이 때, OSP stage 2140 문서의 Real Use Case 그리고 Sequential Diagram을 참조한다.

Retrieve Testable Features

이 때, '쪼갠다' 라는 의미는 module을 함수들로 쪼개는 것과는 또 다른 개념이다. module을 쪼개는 경우는 단순히 개별의 함수들을 떼어 내는 것으로 그 함수가 독립적으로는 수행이 불가능한데 비해 Functional Specification을 Testable Features로 나누는 것은 Testable Feature 그 자체로도 독립적으로 수행 가능한 한 suite로 나누는 것을 말한다.

Retrieve Testable Features

파일

파일 실행

지정한 파일 실행 여부

로그파일 기록 여부

파일 수정

사용자의 임의 입력값 대로 수정하는지 여부

로그파일 기록 여부

Retrieve Testable Features

파일 검색

사용자가 입력한 이름의 파일을 검색하는지 여부

사용자의 옵션 추가 여부

파일 정보 보기

이름, 수정된 날짜, 지정된 태그 등의 정보를 보여주는 여부

Retrieve Testable Features

파일 삭제
의존성 검사 시행 여부
의존성 여부를 사용자에게 알리는지 여부
삭제 알림 여부
로그 파일 기록 여부
파일 삭제 여부

Retrieve Testable Features

태그

태그 생성

중복 이름 검사 여부
태그를 생성하는지 여부
로그 기록 여부

태그 삭제

삭제를 할 태그의 의존성을 검사하는지 여부
해당 태그가 달린 파일 목록을 불러오는지 여부
불러온 파일들에 태그를 삭제하는지 여부
태그를 삭제하는지 여부

Retrieve Testable Features

태그 수정

Tag ID에 해당하는 태그를 받아오는지 여부

태그 정보를 수정하는지 여부

태그 검색

사용자가 입력한 이름의 태그를 찾는지 여부

Retrieve Testable Features

태그 달기

파일에 태그를 다는지 여부

태그 검색

사용자가 입력한 이름의 태그를 찾는지 여부

Retrieve Testable Features

최근 사용한 태그를 저장

최근 태그를 저장하는지 여부

최근 사용한 태그 목록 보기

최근 사용 태그 목록을 불러오는지 여부

Retrieve Testable Features

태그 의존성 해지

선택한 태그의 의존성이 있는지 확인 여부
태그에서 의존성을 제거하는지 여부
의존성을 제거하는지 여부

태그 의존성 부여

부모 관계를 설정하는지 여부
자식 관계를 설정하는지 여부
태그에 의존성을 부여하는지 여부

Retrieve Testable Features

네임태그

네임태그 생성

생성할 네임태그의 이름이 중복되는지 확인 여부
네임태그를 만드는지 여부

네임태그 삭제

네임태그를 삭제하는지 여부

Retrieve Testable Features

네임태그 수정

선택한 네임 태그(객체)를 얻어오는지 여부

중복된 이름인지 확인 여부

네임태그 수정 여부

네임태그 목록보기

네임태그 목록을 불러오는지 여부

Representative Values & Test Cases

Representative Values

Testable Features를 테스트하려면 각 Testable Features에 실제 사용시에 발생할 수 있는 여러 경우의 수를 집어 넣어야 한다. 이 때, 경우의 수도 아무런 값이 아닌 각 Testable Feature가 수행하는 기능에 필요한 값, 말 그대로 기능을 대표할 수 있는 값이 들어 가야한다.
이 값을 Representative Value라고 하며, 여기에서 나올 수 있는 값의 조합으로 실제 test를 돌리는 test case의 조합이 나오게 된다.

Representative Values

본 시스템에서 뽑아낸 Testable Feature에 속하는 각각의 개별적인 기능들은 메소드이고, 각 메소드는 필요한 값을 파라미터로 받고 있음을 보고, 파라미터를 참조하여 Testable Feature의 기능들의 Representative Value를 뽑아냈다.

Representative Value는 한 개 혹은 복수의 값이 나올 수 있다.

Test Cases

Representative Value가 결정된 후에 각 Representative Value들에 들어갈 수 있는 경우의 수를 추려내어 이들의 조합으로 Test Case를 작성하였다.

Representative Value가 3개 이상의 경우, 테스트를 해야하는 경우가 기하급수적으로 늘어나 Pair wise 기법을 이용하여, 2개 요소의 모든 조합을 테스트 한다.

*pair wise 는 테스트를 하는데 필요한 각 값들은 다른 파라미터 값과 최소한 한 번씩은 이룬다는 뜻으로, 테스트를 수행할 때, 대부분의 경우 2가지 요소의 상호작용에 의해 결함이 발견된다는 사실에 착안한다.

Test Cases

파일

파일 실행

executeFile

파일 경로
null
valid

첫 단계에서 시스템을 실행해서
이것저것 눌러보면 기능들을
테스트 해볼 때, 'null' 파일들이
생성되는 현상을 발견하였다.
이 경험을 바탕으로 외부에서 받는
파라미터의 null 여부를 체크했다.
valid는 유효한 값을 의미한다.

Test Cases

파일 삭제
checkDependency

파일 index id
-1
1

파일 인덱스 번호, 고유 번호를 나타내는 '아이디' 값는 곧 배열의 인덱스 번호로 사용되어지므로 음수 값이 나오면 안된다.
그 외의 양수값은 1로 한다.

Test Cases

파일 삭제
removeFile

파일 index id
-1
1

Test Cases

파일 수정 checkDuplication

파일 경로	새 이름
null	null
valid	null
valid	valid
null	valid

Test Cases

파일 수정
renameFile

파일 index id	새 이름
-1	null
1	null
1	valid
-1	valid

Test Cases

파일 검색

searchFile

태그 여부	이름
T	null
F	null
T	valid
F	valid

Test Cases

태그

태그 생성

checkDuplicate

이름
null
valid

Test Cases

태그 생성

makeTag

이름	타입	Color	dependency
null	dependency	red	object
" "	related	yellow	
valid	subtype	green	
	essential	blue	
		black	

타입과 color는 열거형 상수 값으로 테스트를 하는데 영향을 미치지 않고, dependency의 경우 object 값이 들어오게 되는데, Junit으로는 테스트에 한계가 있어 이름으로만 테스트를 한다.
" " 의 경우 white-space 를 의미한다.

Test Cases

태그 삭제

searchTagIndex

태그 ID
-1
1

Test Cases

	태그 ID
태그 삭제 deleteFile	-1
	1

Test Cases

태그 삭제
removeTag

태그 ID
-1
1

Test Cases

	태그 ID
태그 수정	
getTag	-1
	1

Test Cases

태그 수정
modifyTag

ID	이름	타입	Color	dependency
-1	null	dependency	red	object
-1	" "	related	yellow	
-1	valid	subtype	green	
1	null	essential	blue	
1	" "		black	
1	valid			

앞선 경우와 같은 이유로 ID와 이름으로만 테스팅을 한다.

Test Cases

태그 지정
appointTag

ID	경로	태그ID
-1	null	1
-1	valid	-1
1	valid	1
1	null	-1

Pair wise 를 이용할 경우,
각 다른 값에 대해서도 (태그ID-경로 / 태그ID-ID) 겹치는 경우가 생기지 않도록 교차해서 만든다.

Test Cases

태그 해제 isTagged	경로
	null
	valid

Test Cases

태그 해제	경로
clearTag	null
	valid

Test Cases

의존성 해지
getDependency

dep ID
1
-1

Test Cases

의존성 해지	dep ID
deleteDependency	1
	-1

Test Cases

의존성 부여

setParent

태그 ID	Parent ID
1	1
1	-1
-1	1
-1	-1

Test Cases

의존성 부여
setChild

태그 ID	Child ID
1	1
1	-1
-1	1
-1	-1

Test Cases

의존성 부여
addDependency

dep ID
1
-1

Test Cases

네임태그

네임태그 만들기

createNameTag

이름	설명	디렉토리
null	null	null
valid	null	valid
null	valid	valid
valid	valid	null

Test Cases

네임태그 지우기

deleteNameTag

ID
1
-1

Test Cases

네임태그 보기

getNameTag

ID
1
-1

Test Cases

네임태그 수정

modifyNameTag

ID	이름	location	설명
-1	null	null	valid
-1	valid	valid	null
1	null	valid	null
1	valid	null	valid
-1	null	valid	null
-1	valid	null	valid
1	null	null	valid
1	valid	valid	null

Test Cases

기타

로그 기록	로그 문장
writeLog	null
	valid

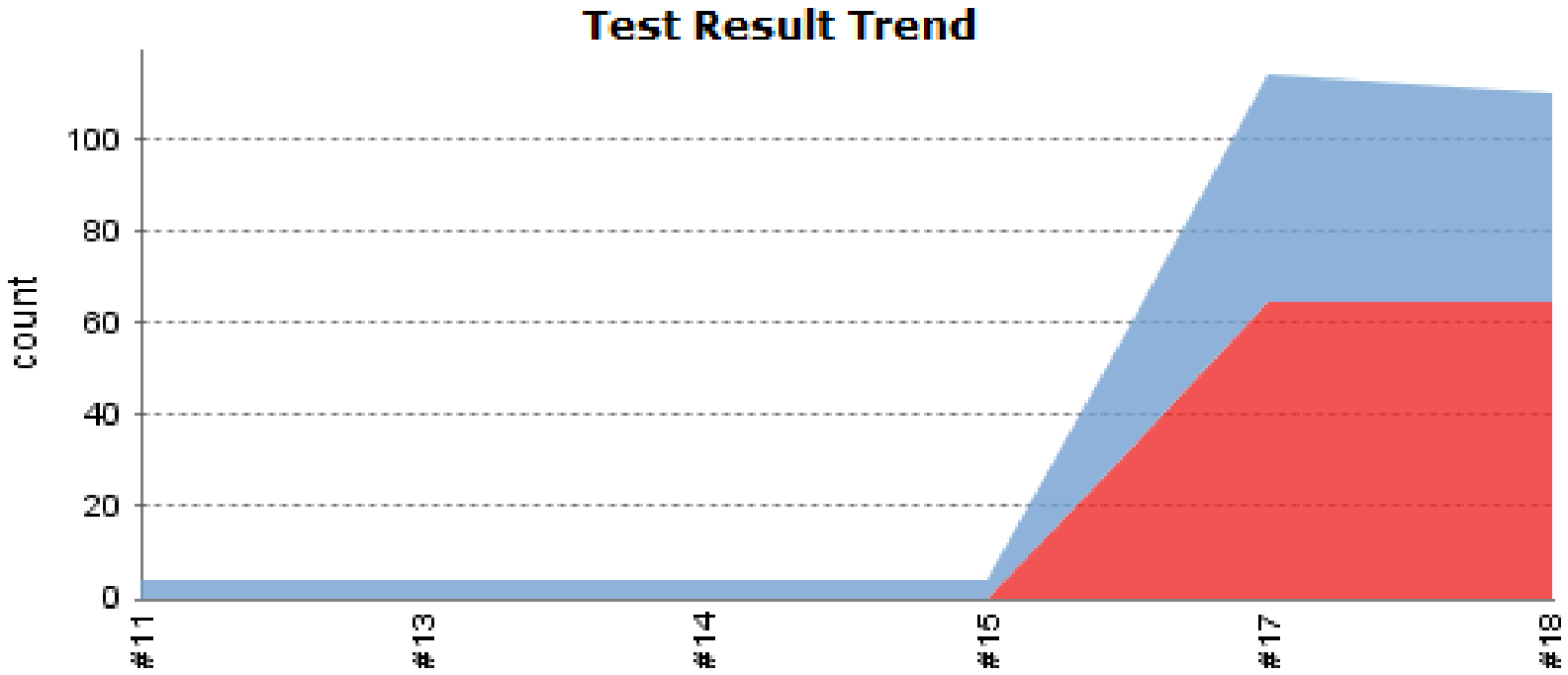
Test Result

Hudson Test Report

All Tests

Class	Duration	Fail	(diff)	Skip	(diff)	Total	(diff)
appointTagTest	60 ms	2		0		8	
clearTagTest	49 ms	2		0		4	
createTagTest	89 ms	8		0		16	
deleteDependencyTest	0.17 sec	4		0		8	
executeFileTest	80 ms	2		0		4	
modifyTagTest	94 ms	4		0		12	
removeFileTest	41 ms	2		0		8	
removeTagTest	31 ms	2		0		4	
renameFileTest	83 ms	16		0		16	
searchFileTest	53 ms	4		0		6	
setDependancyTest	58 ms	16		0		20	
wirteRecentTagTest	99 ms	2		0		4	

Hudson Test Report



JFeature Requirement Test Report

Requirement Coverage Report

Requirement Coverage Summary

Summary (18)	2 (11.11%) / 16 (88.89%)

Number of Requirements	**18**
Unique Test Methods	**62**
Requirements:Test Methods Ratio	**1:3**
Missing Test Methods	**None**
Unmapped Test Methods	**11**

Requirement Coverage Details

Sr#	Category	Coverage
1.	태그 (8)	8 (100%)
2.	파일 (6)	2 (33.33%) / 4 (66.67%)
3.	로그 (1)	1 (100%)
4.	네임태그 (3)	3 (100%)

JFeature Requirement Test Report

태그

Requirement Coverage Details

Sr#	Coverage Item	Coverage
1.	태그 이름과 태그와 관련된 설명 등을 입력받아 태그를 만든다 (6)	3 (50%) / 3 (50%)
2.	입력 받은 이름의 중복을 체크한다 (2)	1 (50%) / 1 (50%)
3.	생성된 태그를 삭제한다 (2)	1 (50%) / 1 (50%)
4.	생성된 파일에 태그를 지정할 수 있다 (4)	3 (75%) / 1 (25%)
5.	파일에 지정된 태그를 해제할 수 있다 (2)	1 (50%) / 1 (50%)
6.	TDMS상에 태그끼리의 의존성을 부여한다 (10)	2 (20%) / 8 (80%)
7.	태그 사이의 의존성을 삭제한다 (2)	1 (50%) / 1 (50%)
8.	최근 사용한 태그를 저장한다 (2)	1 (50%) / 1 (50%)

JFeature Requirement Test Report

파일

Requirement Coverage Details

Sr#	Coverage Item	Coverage
1.	TDMS에서 파일을 실행할 수 있다 (1)	1 (100%)
2.	TDMS에 존재하는 파일을 삭제한다 (2)	1 (50%) 1 (50%)
3.	파일 삭제 시 의존성 검사를 한다. (2)	2 (100%)
4.	TDMS에서 파일 이름을 다시 정할 수 있다 (6)	6 (100%)
5.	중복되는 이름으로 수정할 수 없다 (2)	2 (100%)
6.	TDMS에서 파일 이름으로 검색할 수 있다 (3)	1 (33.33%) 2 (66.67%)

JFeature Requirement Test Report

로그

Requirement Coverage Details

Sr#	Coverage Item	Coverage
1.	TDMS상에서의 파일, 태그에 대한 변경 사항을 로그로 기록한다 (2)	1 (50%) 1 (50%)

JFeature Requirement Test Report

네임태그

Requirement Coverage Details

Sr#	Coverage Item	Coverage
1.	네임 태그를 생성한다 (4)	4 (100%)
2.	네임 태그를 삭제한다 (2)	1 (50%) 1 (50%)
3.	네임 태그를 수정할 수 있다 (8)	8 (100%)

Test Result

테스트 결과의 많은 부분이 예외 처리를 안 한 부분에 대한 것들로, 이 부분을 보완하면 상당 수의 Fail된 테스트에 통과할 것이다.

Jfeature 요구사항의 경우, 요구사항 자체가 만족되지 않았다기 보다는 요구사항을 수행하는 메소드가 있음에도 해당 메소드가 테스트를 통과하지 못해 대다수의 요구사항이 불만족으로 나왔다.

Conclusion

Conclusion

처음 한 테스팅으로 많은 애로 사항이 있었다.

Testable Feature까지 분해하기

구체적으로 어떻게 그리고 얼마만큼 잘게 잘라야 하는지 그리고 정확히 각 용어들이 명시하는 것이 우리가 명세해 놓은 것인지 여부 등, 초기 Testable Feature를 뽑는데 헤매었다.

Junit 테스트의 한계점

Junit의 경우, 특정 조건에 대해서 리턴값과 예상값을 비교하여 같은지 여부로 기능이 제대로 수행되는지 확인을 하게 되는데, 특별한 리턴값이 없는 함수에 대해서는 테스트 하기가 곤란했다.

Conclusion

Junit 테스트의 한계점(2)

단위 테스트를 위해 각 메소드 별로 테스트를 하다 보니, 외부에서 받아오는 파라미터 등의 의존성이 존재할 경우, 임의의 테스트가 불가능했다.

미숙한 Tool 사용 능력

아직도 tool들의 몇몇 부분에 대해서는 사용이 미숙하여 자주 헷갈리곤 했다. 특히나 Junit의 경우, 테스트를 작성해야 함에 있어 그 문법을 충분히 공부를 하지 못해 사용하는데 많은 어려움과 한계를 가졌다.

Team 7

Team 7 test but..

초기 설계와 실제 구현 상이

Inspection 과정에서 설계와 실제 구현이 달라서 코드 테스트 이전에 먼저 설계에 맞게 구현을 재부탁

뷰와 로직이 미분리된 코드

구현된 기능만이라도 functional test를 시도했으나, 구현된 코드가 로직과 뷰가 분리되어 있지 않았고, 문서가 미흡하여 단위 별로 자르기가 불가능하다고 판단, 테스트 시행하지 못함.