Graphical Clone Checker

Team #3
201211341 김태현
201411259 고수창
200911411 이상규

Contents

Stage 1001. Define Draft Plan

1. **Motivation**
   - 시간이 지날수록 학생들이 과제를 표절해서 내는 방법이 발전되고 있다. 저학년 프로그래밍 수업의 경우, 소스 파일의 변수명이나 코드의 순서만 바꿔서 만드는 경우도 허다하다. 사람의 눈으로 이 모든 걸 비교하고 검사하는 일은 사실상 불가능하기 때문에, 해당 문제를 해결하기 위한 프로그램을 개발하기로 하였다.

2. **Project Objectives**
   - 다수의 파일을 비교할 수 있다. 이 때, 다수의 파일 선택은 키보드 특수키와 클릭(Ctrl + 클릭, Shift + 클릭, Ctrl + A)으로 제한한다.
   - Clone Checker의 경우, 비교해야 할 Target이 많다. (1:n)
   - Source Code의 Clone Checker의 경우, Project내에 Source가 분산되어 있으므로 여러 개의 파일을 모두 다 비교해야 한다.
   - 결과를 Percentage로 보여준다.
   - 다수의 파일을 비교하고, 도출되는 결과들을 그래프를 이용하여 표시한다.
   - Percentage를 Y축, Test Filter를 X축으로 하여 꺽은선 그래프로 평균치와 결과치를 비교하여 나타낸다.
   - Clone Checker를 구현하기 위해서는 컴파일러 수준의 구문 분석이 필요하지만 여건 상 문자열 유사도를 구하는 프로그램을 만든다.

3. **Functional Requirement**
   - Open Files
   - Delete Files
   - Display File List
   - Clear
   - Configure
   - Compare
   - Check Raw Text
   - Parse File
   - Check Name
   - Check Comment
   - Check Condition
   - Check Loop
   - Display Result
   - Calculate Scores
   - Make Graph
   - Exit

4. Non-functional Requirement

- 결과를 표시할 때 강조하기 위해 원색을 사용한다.
- 결과를 Percentage로 보여준다.
- 결과치와 평균치를 꺽은선 그래프로 표시한다.

5. Resource Estimation

5.1 Human Efforts (M/M)

- 3M/3M

5.2 Human Resources

- 시니어 프로그래머 2명, 주니어 프로그래머 1명

5.3 Duration

- 3 개월

5.4 Budget (단위: 만 원)

	주니어	시니어	전체
인건비	120	500	620
활동비 (식대, 회의비)	10	10	20
계	130	510	640

6. Other Information

- Future Version
  - 모바일용에서도 사용이 가능하도록 Android, iOS 버전을 개발한다.

Stage 1002. Create Preliminary Investigation Report

1. Alternative Solutions

1.1 기존에 나온 Clone Checker를 사용한다.

- 다수의 파일을 한번에 분석하지 못하고, Source Code가 아닌 Report나 자기소개서 등의 문서에 최적화 되어있다.
- 또한, 일반적으로 자기소개서나 Document 위주의 Clone Chcecker 들은 한번에 하나의 File, 혹은 하나의 Document만 검사하기 때문에 다량의 Programming Project File을 한꺼번에 검사하기에는 시간이 오래 걸린다.

1.2 외주 업체에 개발을 의뢰한다.

- 외주 비용이 상당히 높다

1.3 수동으로 검사한다.

- 다수의 파일을 검사할 때 사람이 할 경우 오류가 있을 확률이 있고, 시간이 오래 걸린다. 시간을 줄이기 위해서 아르바이트를 고용하면 인건비가 상당히 많이 필요하다.

2. Project Justification(Business Demands)

- Cost:
  - 1.1 Freeware 이다. (무료로 이용 가능하다.)
  - 1.2 외주 업체에 비해 개발 비용이 저렴하다.
- Duration: 2016년 1학기 안에 완성해야 한다.
- Risk: OOPT Skill, 직장, 연애, 건강, Programming Skill, Plan

  Management
- Effect:
  - 1.1 한꺼번에 여러 파일들을 GUI 환경에서 Clone Checking 할 수 있다.
  - 1.2 Source Code 전용 Clone Checking을 할 수 있다.

3. Risk Management

Risk	Probability	Significance	Weight

OOPT Skill	4	3	12
Programming Skill	3	5	15
Plan Management	3	4	12
직장	2	5	10
연애	1	5	5
건강	1	5	5

4. Risk Reduction Plan

Risk	Reduction Plan
OOPT Skill	이전 프로젝트 자료 참조 및 교수님과 조교님께 자문을 구한다.
Programming Skill	Programming 관련 서적 및 Official API Document를 참조하여 공부를 한다.
Plan Management	일정이 밀리지 않도록 주기적으로 팀원들간에 피드백을 하는 시간을 갖고 최대한 일정에 맞추려고 노력한다.
직장	야근을 최대한 멀리한다.
연애	사랑 싸움을 최대한 멀리하여 감정적인 소모를 방지한다.
건강	평소에 주기적인 운동과 규칙적인 생활을 한다. 아플 때는 바로 병원에 가도록 한다.

5. Market Analysis

- 대부분의 Clone Checker는 소프트웨어 소스 파일의 표절 여부보다는 논문이나 자기소개서 등 일반적인 문서의 표절 여부를 체크하는 Plagiarism Checker 위주이다.

6. Other Managerial Issues

- Deadline : 2016년 6월

Stage 1003. Define Requirements

1. Functional Requirements

Function	Description
Open Files	비교하고자 하는 파일 열기
Delete Files	선택한 파일들 삭제하기
Display File List	열려있는 파일 목록 보여주기
Clear	파일 목록 전체 제거하기
Configure	몇 퍼센트 이상의 비교 결과만을 표시할지 설정하기
Compare	선택한 파일들 비교하기
Check Raw Text	텍스트 단위의 단순 비교
Parse File	소스 파일 파싱하기
Check Name	변수명, 함수명 변경 검사하기
Check Comment	주석의 내용 검사하기
Check Condition	if - switch 변경 검사하기
Check Loop	for - while 변경 검사하기
Display Result	결과 보여주기
Calculate Scores	검사 결과 점수 계산하기
Make Graph	그래프 그리기
Exit	프로그램 종료하기

2. System Functions

Ref. #	Function	Category
R1.1	**Open Files**	Evident
R1.2	**Delete Files**	Evident
R1.3	**Display File List**	Hidden
R1.4	**Clear**	Evident
R2.1	**Configure**	Evident
R3.1	**Compare**	Evident
R3.2	**Check Raw Text**	Hidden
R3.3	**Parse File**	Hidden
R3.4	**Check Name**	Hidden
R3.5	**Check Comment**	Hidden
R3.6	**Check Condition**	Hidden
R3.7	**Check Loop**	Hidden
R3.8	**Calculate Scores**	Hidden
R4.1	**Display Result**	Hidden
R4.2	**Make Graph**	Hidden
R5	**Exit**	Evident

3. Performance Requirements

- 50개의 C 소스 파일을 검사하는 데 필요한 대기시간이 5분보다 적어야 한다.
- 작동환경에서 동작하여야 한다.

4. Operating Environments

- OS : Microsoft Windows 7 이상
- JRE : JAVA 8

5. Development Environments

	A	B	C

OS	Windows 10	OS X 10.11	OS X 10.11
CPU	Intel Core i7-3517U	Intel Core i5-4258U	Intel Core i5-4258U
Memory	6 GB	8 GB	8GB

6. Interface Requirements

- 누구나 쉽게 사용할 수 있도록 최대한 단순한 UI를 제공한다.
- 결과가 명확하게 확인될 수 있도록 정량적 결과를 적절한 색과 방법으로 제공한다.

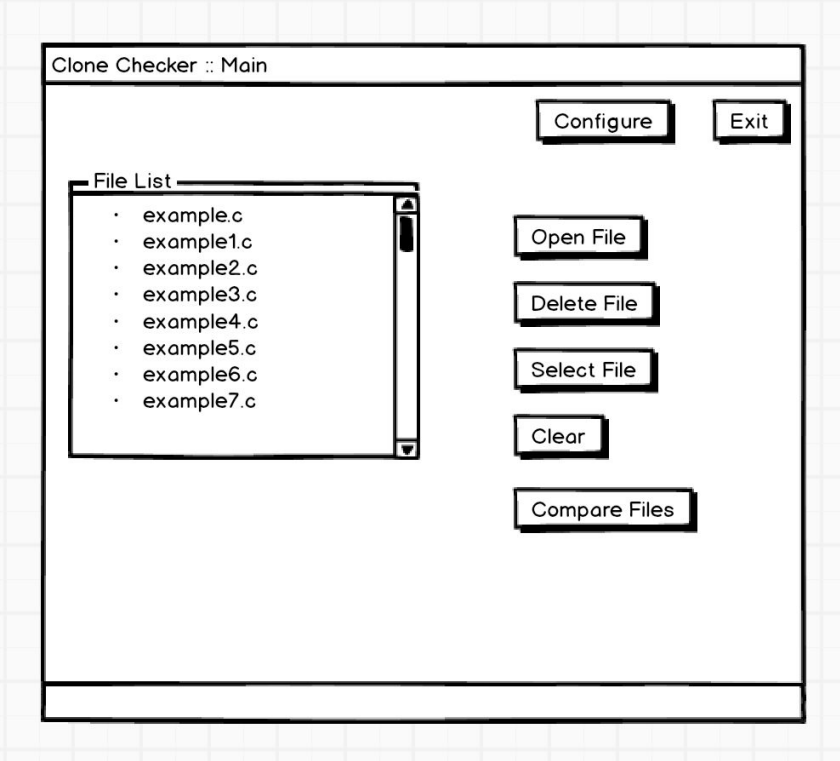
7. Other Requirements

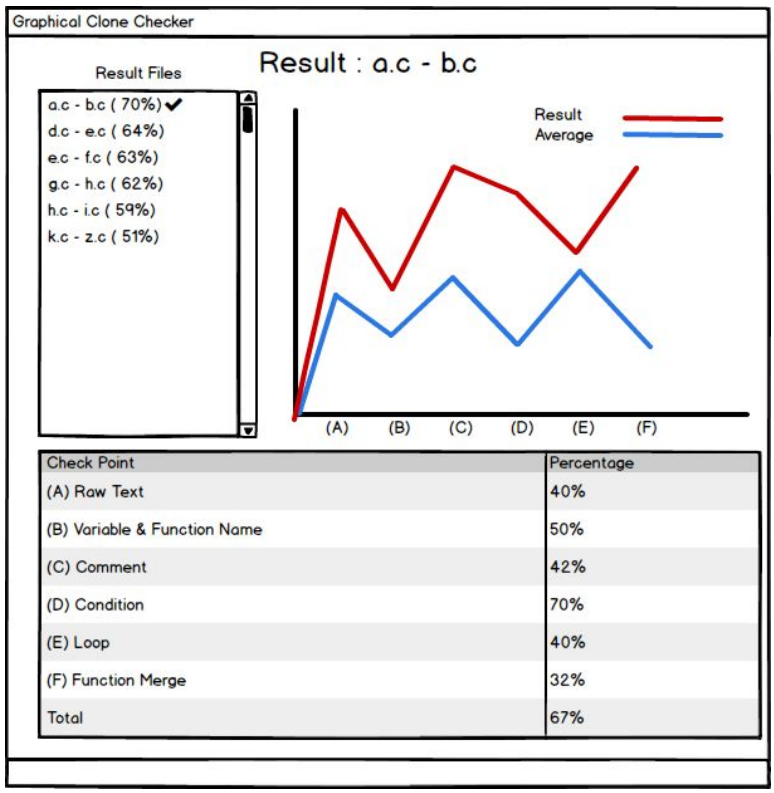
- N/A

Stage 1004. Record Terms in Glossary

Terms	Description	Remarks
File	표절 여부를 확인하고 싶은 소스 파일	
Score	표절 여부의 정량적인 결과 수치	

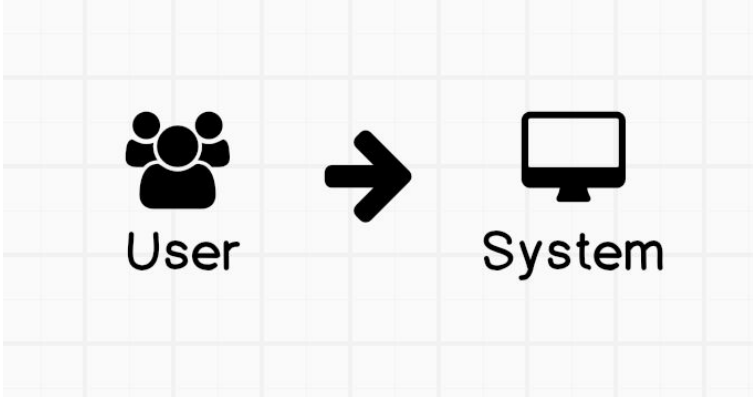
Stage 1005. Implement Prototype





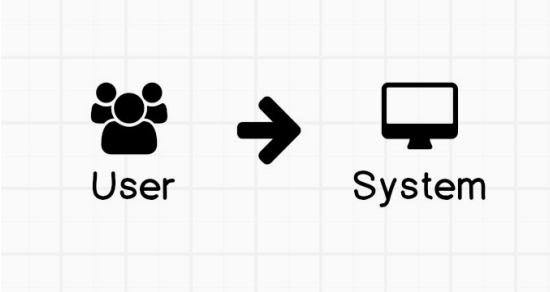
Stage 1006. Define Draft System Architecture

- **Stand-Alone (1 Tier)**



Stage 1007. Define Business Use Case

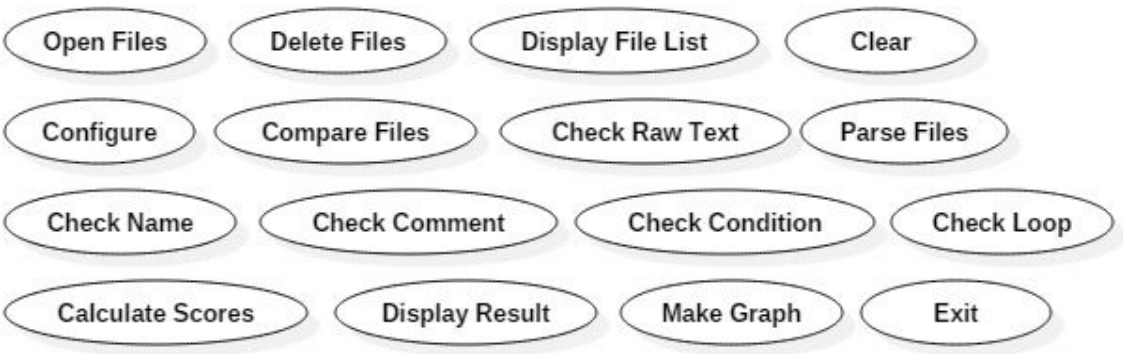
1. **Define System Boundary**



2. **Identify and Describe Actors**
   - User (사용자)
     - 과제의 표절 여부를 검사하려는 교수님 및 조교

3. **Identify Use Case**
   a. actor-based

   b. event-based



4. **Allocate System Functions into Related Use Case**

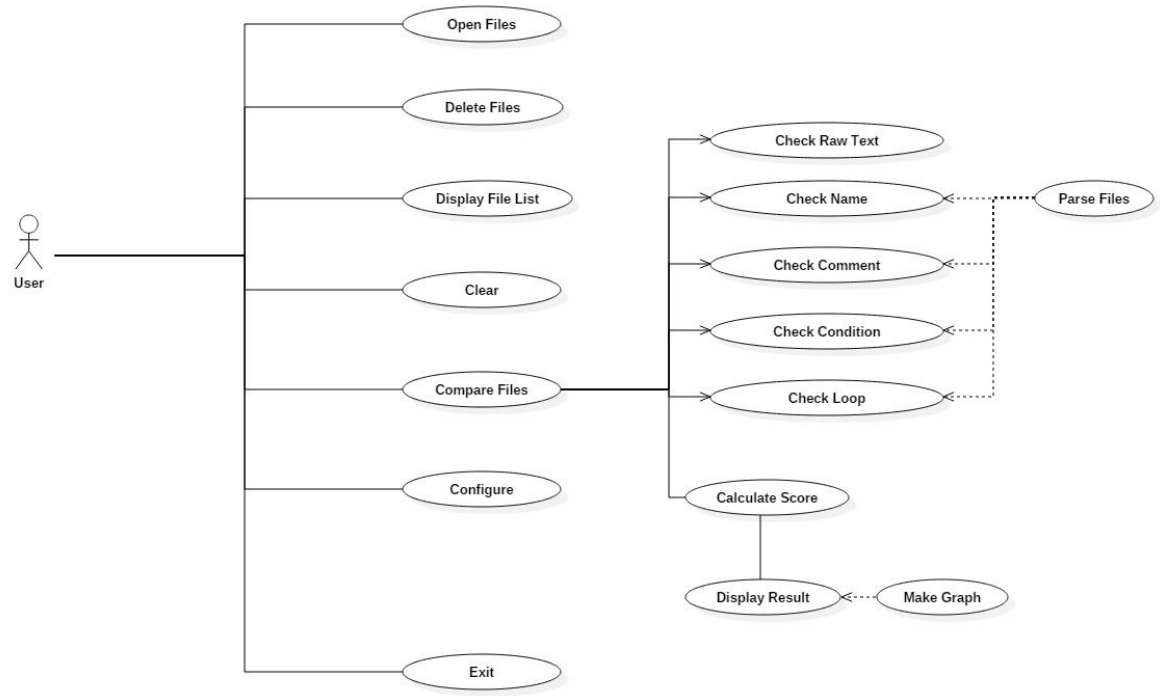
Ref.#	Function	Use Case
R1.1	openFiles	1.Open Files
R1.2	deleteFiles	3.Delete Files
R1.3	displayFileList	4.Display File List
R1.4	clear	5.Clear
R2.1	configure	6.Configure
R3.1	compare	7.Compare
R3.2	checkRawText	8.Check Raw Text
R3.3	parseFile	9.Parse File
R3.4	checkName	10.Check Name
R3.5	checkComment	11.Check Comment
R3.6	checkCondition	12.Check Condition
R3.7	checkLoop	13.Check Loop
R3.8	calculateScores	15.Calculate Scores
R4.1	displayResult	16.Display Result
R4.2	makeGraph	17.Make Graph
R5	exit	19.Exit

5. **Categorize Use Case**

Ref.#	Function	Category
R1.1	**openFiles**	Primary
R1.2	**deleteFiles**	Secondary
R1.3	**displayFileList**	Primary
R1.4	**clear**	Secondary
R2.1	**configure**	Primary
R3.1	**compare**	Primary
R3.2	**checkRawText**	Primary
R3.3	**parseFile**	Primary
R3.4	**checkName**	Primary
R3.5	**checkComment**	Primary
R3.6	**checkCondition**	Primary
R3.7	**checkLoop**	Primary
R3.8	**calculateScores**	Primary
R4.1	**displayResult**	Primary
R4.2	**makeGraph**	Primary
R5	**exit**	Primary

6. **Identify the Relationships Between Use Case**
   - N/A

7. Draw a Use Case Diagram



8. Describe Use Case

- Name : Open Files
  - Actors : User
  - Description :
    - 사용자가 비교하고자 하는 파일들을 Open하는 것이다.
    - 이미 Open한 파일과 동일한 파일명을 가진 파일을 Open할 경우 경고 메시지를 띄운다.
    - Open한 파일은 파일 목록에 파일명을 띄운다.

- Name : Delete Files
  - Actors : User
  - Description :
    - 선택된 파일들을 파일목록에서 삭제한다.
    - 아무 파일도 선택되지 않았을 경우 무시한다.

- Name : Display File List
  - Actors : None
  - Description :
    - Open된 파일 목록을 표시한다.

- Name : Clear
  - Actors : User
  - Description :
    - 파일 목록에 있는 모든 파일들을 목록에서 삭제한다.
    - 목록에 아무것도 없을 경우 무시한다.

- Name : Compare
  - Actors : User
  - Description :
    - 선택된 파일들을 각각 1:1로 비교 분석한다.
    - 아무 파일도 선택되지 않았거나 1개의 파일만 선택되었을 경우 무시한다.
    - 분석이 끝났을 경우 결과를 보여준다.

- Name : Check Raw Text
  - Actors : System
  - Description :
    - 두 파일이 텍스트 단위에서 얼마나 비슷한지 정량적으로 검사한다.

- Name : Parse File
  - Actors : System
  - Description :
    - C 파일을 파싱한다.

- Name : Check Name
  - Actors : System
  - Description :
    - 변수명이나 함수명이 바뀌었는지 정량적으로 검사한다.

- Name : Check Comment
  - Actors : System
  - Description :
    - 주석이 얼마나 비슷한지 정량적으로 검사한다.

- Name : Check Condition
  - Actors : System
  - Description :
    - if문과 switch문이 얼마나 비슷한지 정량적으로 검사한다.

- Name : Check Loop
  - Actors : System
  - Description :
    - for문과 while문이 얼마나 비슷한지 정량적으로 검사한다.

- Name : Calculate Scores
  - Actors : System
  - Description :
    - 검사 결과를 전체적으로 계산한다.

- Name : Display Result
  - Actors : System
  - Description :
    - 결과를 보여준다.

- Name : Make Graph
  - Actors : System
  - Description :
    - 결과를 토대로 그래프를 작성한다.

- Name : Configure
  - Actors : User
  - Description :
    - 몇 퍼센트 이상의 결과를 보여줄 것인지 설정한다.
    - 확인 버튼과 취소 버튼이 있다.
    - 확인 버튼을 누르면 설정 정보가 저장되어 그 다음 결과부터 반영이 된다.
    - 취소 버튼을 누르면 설정 정보가 저장되지 않는다.

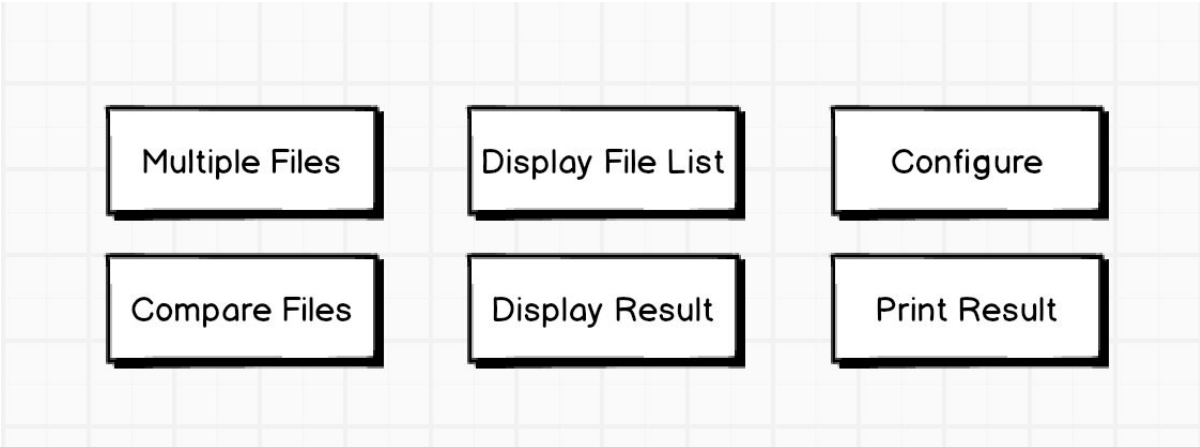
- Name : Exit
  - Actors : User
  - Description :
    - 프로그램을 종료한다.

9. Rank Use Case

Rank	Use Case
High	**Open Files**

Medium	**Delete Files**
Medium	**Display File List**
Medium	**Clear**
Medium	**Configure**
High	**Compare**
High	**Check Raw Text**
High	**Parse File**
High	**Check Name**
High	**Check Comment**
High	**Check Condition**
High	**Check Loop**
High	**Calculate Scores**
High	**Display Result**
High	**Make Graph**
Medium	**Exit**

Stage 1008. Define Business Concept Model



Stage 1009. Define System Test Case

Ref.#	Function	Test Case
R1.1	**Open Files**	- Clone check할 대상 파일이 잘 열리는지 확인한다. - Open한 파일이 파일 목록에 잘 뜨는지 확인한다.
R1.2	**Delete Files**	- 선택한 파일을 삭제했을 때 파일 목록에서 없어졌는지 확인한다. - 다수의 파일을 한꺼번에 삭제 가능한지 확인한다.
R1.3	**Display File List**	- 파일들이 파일 목록에 뜨는지 확인한다.
R1.4	**Clear**	- Clear를 눌렀을 때 모든 파일이 파일 목록에서 잘 사라졌는지 확인한다.
R2.1	**Configure**	- 설정한 값이 결과에 잘 적용되는지 확인한다. - 설정값이 음수이거나 100을 넘어갔을 때 경고창이 뜨는지 확인한다.
R3.1	**Compare**	- 파일 목록에 파일이 없거나 하나밖에 없을 경우에 경고창이 뜨는지 확인한다. - 파일 비교가 제대로 되는지 확인한다.
R3.2	**Check Raw Text**	- Raw text 비교가 제대로 되는지 확인한다.
R3.3	**Parse File**	- Source code의 parsing이 제대로 되는지 확인한다.
R3.4	**Check Name**	- 함수명이나 변수명만 바꾸었는지를 잘 찾아내는지 확인한다.
R3.5	**Check Comment**	- 주석이 얼마나 비슷한지를 잘 찾아내는지 확인한다.
R3.6	**Check Condition**	- if-switch문을 변경한 것을 잘 찾아내는지 확인한다.
R3.7	**Check Loop**	- for-while을 변경한 것을 잘 찾아내는지 확인한다.
R3.8	**Calculate Scores**	- 결과 점수가 잘 계산되는지 확인한다.
R4.1	**Display Result**	- 결과 목록 및 그래프가 잘 작성되는지 확인한다.

R4.2	**Make Graph**	- 그래프가 잘 작성되는지 확인한다.
R5	**Exit**	- 프로그램이 정상적으로 종료되는지 확인한다.

Stage 1010. Refine Plan

1. **Project Scope**
   - 다수의 C 소스 파일을 비교하여 정량적인 기준으로 Cheating 여부를 확인할 수 있다.
   - 결과를 강조색, Percentage, 그래프를 이용하여 표시한다.

2. **Project Objectives**
   - 다수의 파일을 비교할 수 있다.
   - Clone Checker의 경우, 비교해야 할 Target이 많다. (1:n)
   - Source Code의 Clone Checker의 경우, Project내에 Source가 분산되어 있으므로 여러 개의 파일을 모두 다 비교해야 한다.
   - 결과를 Percentage로 보여준다.
   - 다수의 파일을 비교하고, 도출되는 결과들을 그래프를 이용하여 표시한다.
   - Percentage를 Y축, Test Filter를 X축으로 하여 꺽은선 그래프로 평균치와 결과치를 비교하여 나타낸다.
   - Clone Checker를 구현하기 위해서는 컴파일러 수준의 구문 분석이 필요하지만 여건 상 문자열 유사도를 구하는 프로그램을 만든다.

3. **Functional Requirements**

Function	Description
Open Files	비교하고자 하는 파일 열기
Delete Files	선택한 파일들 삭제하기
Display File List	열려있는 파일 목록 보여주기
Clear	파일 목록 전체 제거하기
Configure	몇 퍼센트 이상의 비교 결과만을 표시할지 설정하기
Compare	선택한 파일들 비교하기
Check Raw Text	텍스트 단위의 단순 비교
Parse File	소스 파일 파싱하기
Check Name	변수명, 함수명 변경 검사하기
Check Comment	주석의 내용 검사하기
Check Condition	if - switch 변경 검사하기
Check Loop	for - while 변경 검사하기

Display Result	결과 보여주기
Calculate Scores	검사 결과 점수 계산하기
Make Graph	그래프 그리기
Exit	프로그램 종료하기

4. **Performance Requirements**
   - 50개의 C 소스 파일을 검사하는 데 필요한 대기시간이 5분보다 적어야 한다.
   - 작동환경에서 동작하여야 한다.

5. **Operating Environments**
   - OS : Microsoft Windows 7 이상
   - JRE : JAVA 8

6. **User Interface Requirements**
   - 누구나 쉽게 사용할 수 있도록 최대한 단순한 UI를 제공한다.
   - 결과가 명확하게 확인될 수 있도록 정량적 결과를 적절한 색과 방법으로 제공한다.

7. **Other Requirements**
   - N/A

8. **Recources**

- Human Efforts (M/M)
  - 3M/3M

- Human Resources
  - 시니어 프로그래머 2명, 주니어 프로그래머 1명

- Duration
  - 3 개월

- Budget (단위: 만 원)

	주니어	시니어	전체
인건비	120	500	620
활동비 (식대, 회의비)	10	10	20
계	130	510	640

9. Scheduling

Stage	Phase(00x0)/Activity(000x)	Schedule(Week) 1	2	3	4	5	6	7	8	9	10
1000. Plan & Elaborate	1001. Define Draft Plan	■									
	1002. Create Preliminary Investigation Report	■									
	1003. Define Requirements	■	■								
	1004. Record Terms in Glossary		■								
	1005. Implement Prototype		■	■							
	1006. Define Use Cases			■							
	1007. Define Draft Conceptual Model			■							
	1008. Define Draft System Architecture			■							
	1009. Refine Plan			■	■						
2000. Build	2010. Revise Plan				■	■					
	2020. Synchronize Artifacts					■	■				
	2030. Analyze										
	2031. Define Essential Use Case					■					
	2032. Refine Use Case Diagrams					■					
	2033. Refine Conceptual Model					■					
	2034. Refine Glossary					■					
	2035. Define System Sequence Diagrams					■	■				
	2036. Define Operation Contracts						■				
	2037. Define State Diagrams						■				
	2040. Design										
	2041. Define Real Use Cases						■				
	2042. Define Reports, UI and Storyboards						■				
	2043. Refine System Architecture						■				
	2044. Define Interaction Diagrams						■				
	2045. Define Design Class Diagrams						■	■			
	2046. Define Database Schema							■			
	2050. Construct										
	2051. Implement Class & Interface Definition							■			
	2052. Implement Methods.							■			
	2053. Implement Windows								■		
	2054. Implement Reports								■		
	2055. Implement DB Schema								■		
	2056. Write Test Code								■		
	2060. Test										
	2061. Unit Testing									■	
	2062. Integration Testing									■	
	2063. System Testing										■
	2064. Performance Testing										■
	2065. Acceptance Testing										■
	2066. Documentation Testing										■