논문 2026-2-5 http://dx.doi.org/10.29056/jsf.2026.06.05

역컴파일을 이용한 C# 이진파일의 감정 사례 연구

천준석*, 김영훈*, 우균**†

A Case Study on Software Appraisal of C# Binaries Using Reverse Compilation

Junseok Cheon*, Yeonghun Kim*, Gyun Woo**†

요 약

소프트웨어 감정에서 원본 소스코드 없이 이진파일—실행 파일이나 DLL(dynamic link libraries)—만 존재하면 소스코드 유사도 분석을 직접 적용하기 어렵다. 이 논문은 역컴파일을 통해 C# 이진파일 쌍을 감정하는 사례 연구를 소개한다. 파일 형식과 실행 환경 확인 후 이진파일에 역컴파일을 적용하였다. 복원 코드는 두 가지 방법(정량 분석 및 정성 분석)을 이용하여 비교하였는데, 정성 분석은 문자열 리터럴 및 제어 구조 비교를 포함한다. 전체 유사도는 80.1%에 달하는 것으로 나타났는데, 이를 통해 C# 이진파일 비교에 관한 역컴파일 기법의 효과를 확인할 수 있다. 역컴파일 기법은 하이브리드 구현을 이용하는 다른 언어에도 적용될 수 있을 것이다.

Abstract

When only binary files—executables or DLLs (dynamic link libraries)—are available without the original source code in software appraisal, conventional source-code similarity analysis is difficult to apply directly. This paper introduces a case study to measure the similarity of a pair of C# binaries using reverse compilation. Identifying the file formats and runtime environments, the reverse compilation is applied to the binary files. The recovered source codes are compared in two ways: quantitative and qualitative analyses. The latter encompasses comparisons of string literals and control structures. The overall similarity is revealed up to 80.1%, which shows the effect of the reverse compilation on comparing C# binaries. The reverse compilation method can also be applied in other languages, adopting the hybrid implementation method.

한글키워드 : C# 이진파일, 역컴파일, 소프트웨어 유사도, 소프트웨어 감정, 코드 클론 탐지, 소프트웨어 포렌식

keywords : C# binaries, reverse compilation, software similarity, software appraisal, code clone detection, software forensics

1. 서 론

소프트웨어 감정은 저작권 분쟁 등에서 프로그램 사이의 구현상 공통점을 객관적으로 확인하는 데 활용된다[1]. 상용 소프트웨어에는 개발자

* 부산대학교 정보융합공학과
** 부산대학교 정보컴퓨터공학부
† 교신저자: 우균(email: woogyun@pusan.ac.kr)
접수일자: 2026.06.01. 심사완료: 2026.06.10.
게재확정: 2026.06.20.

의 구현 방식과 업무 지식뿐 아니라 데이터베이스 구조와 데이터 처리 절차 등이 반영된다. 이러한 구현 요소는 같은 기능을 수행하는 프로그램이라도 개발자와 개발 조직에 따라 다르게 나타나는 것이 일반적이기 때문이다.

소스코드가 확보된 경우에는 기존의 코드 유사도 분석 방법을 적용할 수 있다. 기존 연구에서는 텍스트와 토큰뿐 아니라 구문 트리 또는 프로그램 의존 구조를 이용하여 동일하거나 유사한 코드 구간을 탐지해 왔다[2-4]. 최근의 코드 유사도 분석 연구는 표면적인 문장 일치를 넘어 코드의 구문적·구조적 공통점을 분석하는 방향으로 확장되어 왔다[5].

그러나 원본 소스코드 없이 실행파일과 DLL(dynamic link library)만 제공되는 경우에는 기존의 소스코드 유사도 분석 방법을 직접 적용하기 어렵다. 기존 방법은 비교할 소스코드가 확보되어 있음을 전제로 하기 때문이다. 또한 동일한 업무를 수행하는 프로그램은 화면 구성이나 기능 흐름이 유사할 수 있으므로, 실행 결과만으로 내부 구현의 유사도를 판단하기 어렵다.

이진파일만 제공된 경우에는 역컴파일을 통해 내부 코드를 소스코드 수준으로 복원할 수 있다. 역컴파일은 이진파일에 포함된 코드를 사람이 분석할 수 있는 소스코드 형태로 재구성하는 과정이다. 특히 C# 프로그램은 CIL(common intermediate language)과 메타데이터를 포함하므로 네이티브 바이너리보다 코드 구조를 상대적으로 쉽게 복원할 수 있다[6].

그러나 단순히 이진파일을 역컴파일하는 것만으로는 코드 유사도 감정을 수행하기 어렵다. 역컴파일 결과에는 외부 라이브러리와 자동 생성 코드가 포함될 수 있으며, 원본과 비교본의 복원 파일이 항상 일대일로 대응하는 것도 아니다. 또한 역컴파일 결과는 원본 소스코드와 완전히 동일하지 않을 수 있으므로, 분석 결과를 해석할 때 이러한 특성을 고려해야 한다[7].

본 연구에서는 이러한 문제를 고려한 역컴파일 기반 감정 절차를 C# 이진파일 감정 사례에 적용한다. 각 이진파일에서 복원된 코드를 비교 가능한 단위로 구성하고, 복원 코드의 유사도를 정량적으로 산출한다. 또한 문자열, SQL 질의문 및 제어 구조 등의 구현 특징을 정성적으로 검토하여 정량 분석 결과를 함께 해석한다.

본 논문의 구성은 다음과 같다. 2장에서는 코드 유사도와 이진파일 분석에 관한 관련 연구를 검토하고, 3장에서는 감정 대상 프로그램과 분석 범위를 제시한다. 4장에서는 역컴파일, 전처리, 대응 범위 선정 및 유사도 분석 절차를 설명하며, 5장에서는 정량 분석과 정성 분석 결과를 제시한다. 6장에서는 분석 결과를 고찰하고, 마지막으로 7장에서 결론을 맺는다.

2. 관련 연구

소스코드 유사도 연구는 토큰 기반 비교에서 구문 및 의미 정보를 분석하는 방법으로 발전해 왔다. Kamiya 등은 소스코드를 토큰열로 변환하여 대규모 코드에서 클론을 탐지하는 CCFinder를 제안하였다[2]. Wu 등은 추상 구문 트리의 노드 전이 관계를 마르코프 연쇄로 표현하여 의미적 코드 클론을 탐지하는 Amain을 제안하였다[8]. 최근 Dou 등은 구문 유형에 따라 분류한 토큰과 대조 학습을 결합하여 단순 클론의 탐지 효율과 의미적 클론의 탐지 능력을 함께 고려한 CC2Vec을 제안하였다[9]. 이들 소스코드 유사도 연구는 이진파일 비교에 직접 활용될 수 없다는 한계가 있다.

이진파일 유사도 연구는 함수와 명령어의 구조적 비교에서 실행 의미와 취약 코드 탐색을 고려하는 방향으로 확장되어 연구되었다. Yang 등은 패치로 변경된 코드의 실행 의미를 분석하여

바이너리에 취약점 패치가 적용되었는지를 확인하는 방법을 제안하였다[10]. Yang 등은 다른 연구에서 도메인 지식에 기반한 사전 필터링과 재순위화를 딥러닝 기반 유사도 분석에 결합한 Asteria-Pro를 제안하였다[11]. Shirani 등은 서로 다른 명령어 집합과 컴파일 환경에서 생성된 바이너리 간의 유사 코드를 탐지하기 위한 교차 아키텍처 분석 방법인 BinX를 제안하였다[12]. 직접 이진파일을 비교하는 이러한 방법은 정확도가 떨어진다는 단점이 있다.

한편, 역컴파일을 소프트웨어 포렌식에 활용하는 연구도 이루어진 바 있는데, Ekanem 등은 Visual Basic 이진파일을 역컴파일하여, 소프트웨어 침해 과정에서 변경된 부분을 확인할 수 있음을 보였다[13]. Ekanem의 연구는 역컴파일을 이용하여 소프트웨어 침해 여부를 분석할 수 있음을 제시하였다.

본 연구는 소스코드가 제공되지 않은 C# 이진파일의 감정을 위해 역컴파일이 사용될 수 있음을 실제 사례를 통해 검증한다. 기존의 소스코드 유사도 연구는 분석 대상의 소스코드가 확보되어 있음을 전제로 하며, 이진파일 유사도 연구는 주로 구조적 비교에 초점을 두었다. 이에 본 연구에서는 역컴파일로 복원한 코드의 유사도와 구현 특징을 함께 분석하여 실제 감정에 활용할 수 있는지를 살펴본다.

3. 감정 대상 및 분석 범위

3.1 감정 대상 프로그램

본 연구의 감정 대상은 C# 기반 POS(point of sale) 프로그램 한 쌍이며, 원본을 A, 비교본을 B로 지칭한다. 두 프로그램은 판매자를 위한 사용자 인터페이스를 제공하고, 상품, 판매, 재고, 고객 및 배송 등의 업무 데이터를 처리한다. 본 감정에서는 A와 B 사이의 코드 및 구현상 공통점을 기술적으로 분석한다.

A와 B는 각각 21개의 이진파일이 분석 대상으로 제공되었다. A는 실행파일 1개와 DLL 20개로 구성되며, B는 실행파일 2개와 DLL 19개로 구성된다. 각 DLL은 프로그램의 주요 업무 기능을 모듈별로 구현한다.

3.2 분석 범위 선정

분석 범위는 두 프로그램의 업무 기능을 직접 구현한 자체 작성 코드로 한정하였다. 자체 작성 여부는 어셈블리 구성, 파일명, 디렉터리와 네임스페이스, 담당 기능, 참조 관계 및 복원 코드의 내용을 종합하여 판단하였다. 데이터 처리, 상품, 판매, 재고, 고객 및 배송 등의 기능을 구현한 코드는 분석 대상에 포함하였다.

외부 라이브러리와 자동 생성 코드는 프로그램 고유의 구현을 나타내지 않으므로 분석 대상에서 제외하였다. RemObjects[1]와 DevExpress[2] 등의 외부 라이브러리, Properties 디렉터리의 자동 생성 코드, 설정 파일 및 리소스 파일은 비교 범위에 포함하지 않았다. 구체적인 분석 대상 선정 기준과 처리 방법은 표 1과 같다.

표 1. 분석 대상 선정 기준
Table 1. Criteria for Selecting Analysis Targets

구분	판단 기준	처리 방법
포함	프로그램 고유 기능을 구현하는 이진파일	대응 파일 선정 후 복원 소스코드 비교
	정확히 대응되는 비교본 파일이 없는 이진파일	비교본의 복원 코드 전체에서 유사 파일 탐색
제외	외부 라이브러리, 자동 생성 코드 및 리소스 파일	유사도 산출 대상에서 제외

1) https://www.remobjects.com/

2) https://www.devexpress.com/

4. 감정 절차 및 분석 방법

4.1 전체 감정 절차

전체 감정 절차는 파일 형식 확인, 역컴파일, 분석 대상 정제, 대응 파일 선정, 정량 및 정성 분석의 순서로 수행했다. 각 단계에서 생성된 결과는 다음 단계의 입력으로 사용하여 분석 대상을 구체화했다. 전체 감정 절차는 그림 1과 같다.

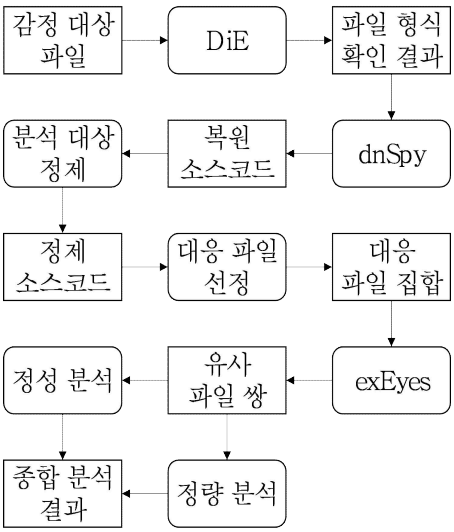

그림 1. 역컴파일 기반 감정 절차
Fig. 1. Reverse Compilation-Based Appraisal Procedure

감정 대상 파일은 DiE[14]를 이용하여 파일 형식과 실행 환경을 확인하고, dnSpy[15]를 이용하여 소스코드 수준으로 복원한다. 이후 분석 대상 정제 및 대응 파일 선정을 거쳐 대응 파일 집합을 구성한다. exEyes[16]는 정제된 대응 파일 집합을 비교하여 유사 파일 쌍과 유사 라인을 도출하며, 이를 기반으로 정량 분석과 정성 분석을 수행한다. 마지막으로 두 분석 결과를 종합하여 프로그램 사이의 코드 수준 공통점을 해석한다.

4.2 역컴파일 및 전처리

분석 대상 파일의 형식과 실행 환경은 DiE 3.10을 이용하여 확인하였다. 확인 결과 두 프로그램의 이진파일은 Windows PE32 형식이며, .NET 프레임워크(버전 4.5.2)와 CLR(common language runtime, 버전 4.0.30319)을 사용하는 것으로 나타났다. 이러한 파일 특성을 바탕으로 각 이진파일의 내부 구조를 소스코드 수준으로 복원할 수 있다고 판단하였다.

각 실행파일과 DLL은 dnSpy(버전 6.1.8)를 이용하여 개별적으로 역컴파일하였다. 하나의 이진파일에서 복원된 여러 소스 파일은 해당 이진파일별 디렉터리에 저장하였다. 복원 과정에서 생성된 하위 구조와 네임스페이스를 유지하여 각 파일의 소속과 기능을 확인할 수 있도록 하였다.

복원 결과는 표 1의 선정 기준에 따라 분석 대상만 남기도록 정제하였다. 외부 라이브러리, 자동 생성 코드, 설정 파일 및 리소스 파일을 제외하고 프로그램의 업무 기능을 직접 구현한 코드만 비교 대상으로 사용하였다. 또한 역컴파일 도구에 따른 비대칭적인 변환을 줄이기 위해 두 프로그램에 동일한 dnSpy 버전과 설정을 적용하였으며, 이를 통해 도구 차이에서 비롯되는 오탐 가능성을 줄였다.

4.3 대응 파일 선정

A와 B 간의 파일 대응 관계는 두 프로그램에서 동일하거나 유사한 업무 기능을 담당하는 이진파일을 기준으로 설정하였다. 이진파일별 복원 디렉터리를 비교 단위로 사용하고, 파일명, 네임스페이스, 담당 기능, 참조 관계 및 복원 코드의 내용을 종합하여 대응 관계를 판단하였다. 동일한 기능을 구현하는 모듈이 확인되면 A와 B의 해당 디렉터리를 일대일로 대응시켰다.

대부분의 이진파일은 파일명과 모듈 구성이 유사하여 직접 대응 관계를 설정할 수 있었다.

데이터 처리, 상품, 판매, 재고, 고객 및 배송 등의 업무 모듈은 동일하거나 유사한 명칭과 기능을 기준으로 대응 파일을 선정하였다. 이렇게 선정된 대응 파일 집합은 이후 정량 분석과 정성 분석의 공통 비교 범위로 사용하였다.

하지만 A의 세 번째 이진파일(A03)은 B에서 직접 대응하는 단일 파일을 확인하기 어려웠다. 관련 구현이 B의 여러 모듈에 분산되어 있을 가능성을 고려하여 A03의 복원 코드 비교 범위는 B 전체로 설정하였다. 이를 통해 파일 구성의 차이로 인해 유사한 구현이 비교 대상에서 누락될 가능성을 줄였다.

4.4 정량 유사도 분석

정량 유사도 분석은 exEyes(버전 5.1, 64-bit)를 이용하여 대응 파일 집합 사이의 유사 라인을 탐지하는 방식으로 수행하였다. exEyes는 소스코드의 키워드 벡터를 기준으로 파일 간 유사도를 비교하고, 유사한 코드 구간과 라인 수를 산출한다. exEyes 설정값은 기본 설정값(유사 판정 3줄 이상, 유사율 80%, 최대토큰비 2.0)[3]을 적용하였다.

exEyes는 C#을 직접 지원하지 않으므로 C, C++, Java 계열의 키워드 설정을 적용했다. 이들 언어와 C#은 주요 예약어 및 키워드가 상당 부분 공통되므로 키워드 설정의 차이가 유사도 산출에 미치는 영향이 제한적인 것으로 판단하였다.[4]

유사도 산출에서는 빈 줄과 공백 문자만 포함된 줄을 제외하고, 주석을 포함한 나머지 코드를 분석 대상으로 사용하였다. 원본 A의 복원 파일 집합($\mathbf{A}$)과 비교본 B의 복원 파일 집합($\mathbf{B}$)은 식 1과 같이 정의한다.

$$\mathbf{A} = \{A_1, A_2, \ldots, A_i, \ldots, A_n\}$$
$$\mathbf{B} = \{B_1, B_2, \ldots, B_j, \ldots, B_m\} \quad (1)$$

두 소스코드 집합의 유사도는 개별 파일의 유사도로 결정되는데, 구체적으로 원본의 개별 파일(A_i)과 가장 유사한 비교본 파일($\widehat{B_i} = B_j$)을 찾는다. 이 작업이 완료되면 원본 기준 전체 유사도 $S(\mathbf{A},\mathbf{B})$는 식 2와 같이 산출할 수 있다.

$$S(\mathbf{A},\mathbf{B}) = \frac{\sum_{i=1}^{n} \left| A_i \cap \hat{B}_i \right|}{\sum_{i=1}^{n} |A_i|} \quad (2)$$

식 2에서 $|A_i|$는 A_i의 분석 라인 수를 의미하며 $\left|A_i \cap \widehat{B_i}\right|$는 파일 A_i와 $\widehat{B_i}$의 유사 라인 수를 의미한다.

4.5 정성적 구현 특징 분석

정성 분석은 정량 유사도만으로 확인하기 어려운 구현상의 공통점을 검토하기 위해 수행하였다. exEyes에서 탐지된 유사 파일 쌍과 코드 구간을 중심으로 두 프로그램의 복원 코드를 대조하였다. 분석에서는 모든 공통점의 빈도를 산출하기보다 프로그램 고유의 구현 방식을 보여주는 대표 사례를 선정하였다.

정성 분석 항목은 문자열, SQL 질의문 및 제어 구조이다. 문자열은 내용뿐 아니라 공백과 구두점의 배치 등을 비교하였으며, SQL 질의문은 사용된 테이블과 필드, 조건절 및 질의문 구성 순서를 검토하였다. 제어 구조는 조건식의 내용과 검사 순서, 분기 및 중첩 구조를 비교하였다.

정성 분석 결과는 정량 유사도의 해석을 보완

3) 3토큰 이상으로 구성된 라인 중 토큰비 차이가 2.0 이내인 라인만 유사도 산정에 고려하며, 80% 이상 유사한 라인을 유사 라인으로 판정하되 3라인 이상 연속으로 유사한 경우만 유사 블록에 포함시키는 설정이다.

4) 추가로 복원 코드를 조사한 결과, C# 특화 키워드(delegate, stackalloc 등)는 거의 발견할 수 없었다.

하는 근거로 사용하였다. 개별 구현 특징만으로는 프로그램 사이의 공통점을 충분히 판단하기 어렵고, 특히 제어 구조는 컴파일과 역컴파일 과정에서 표현이 달라질 수 있기 때문이다. 따라서 문자열, SQL 질의문 및 제어 구조에서 확인된 여러 특징을 정량 유사도와 함께 해석하였다.

5. 분석 결과

5.1 정량 유사도 분석 결과

각 파일의 분석 라인 수, 유사 라인 수 및 A와 B를 각각 기준으로 산출한 유사도는 표 2와 같다. 파일별 정량 분석 결과는 대부분의 대응 범위에서 높은 유사도를 보였다. A03은 직접 대응하는 단일 파일이 없어 4.3절에서 설정한 B의 전체 복원 코드를 비교 범위로 적용하였다.

표 2. 파일별 유사도 산출 결과
Table 2. File-Level Similarity Calculation Results

순번	원본	비교본	LoC_A	LoC_B	유사 라인	S_A (%)	S_B (%)
1	A01.exe	B01.exe	5,637	5,834	5,349	94.9	91.7
2	A02.dll	B03.dll	12,807	11,379	9,825	76.7	86.3
3	A03.dll	B복수파일	12,228	28,321	8,534	69.8	30.1
4	A04.dll	B04.dll	87,117	96,804	65,003	74.6	67.1
5	A05.dll	B05.dll	10,157	8,921	7,323	72.1	82.1
6	A06.dll	B06.dll	3,177	2,819	2,602	81.9	92.3
7	A07.dll	B07.dll	34,628	33,323	28,133	81.2	84.4
8	A08.dll	B08.dll	170,176	166,891	136,079	80.0	81.5
9	A09.dll	B09.dll	207,156	199,662	168,518	81.3	84.4
10	A10.dll	B10.dll	60,158	59,553	49,632	82.5	83.3
11	A11.dll	B11.dll	68,961	67,423	58,418	84.7	86.6
12	A12.dll	B12.dll	150,273	154,249	118,564	78.9	76.9
13	A13.dll	B13.dll	192,909	189,973	152,685	79.1	80.4
14	A14.dll	B14.dll	29,511	28,130	23,764	80.5	84.5
15	A15.dll	B15.dll	93,557	95,143	72,990	78.0	76.7
16	A16.dll	B16.dll	56,991	57,023	47,292	83.0	82.9
17	A17.dll	B17.dll	131,804	128,662	104,072	79.0	80.9
18	A18.dll	B18.dll	46,622	47,571	38,002	81.5	79.9
19	A19.dll	B19.dll	19,198	30,950	14,490	75.5	46.8
20	A20.dll	B20.dll	130,035	133,109	108,642	83.5	81.6
21	A21.dll	B21.dll	4,895	4,542	4,480	91.5	98.6
계			1,527,997	1,550,282	1,224,397	80.1	79.0

표 2에서 확인할 수 있듯이, 전체 분석 대상에서 탐지된 유사 라인은 1,224,397줄이었다. 빈 줄을 제외한 전체 분석 라인은 A가 1,527,997줄이고 B가 1,550,282줄로 확인되었다. 이에 따라 전체 유사도는 A를 기준으로 80.1%, B를 기준으로 79.0%로 산출되었다.

A03은 비교 범위의 차이로 인해 A와 B를 기준으로 한 유사도 사이에 큰 차이를 보였다. A03의 분석 라인은 12,228줄이고 B의 비교 범위는 28,321줄이며, 이 가운데 8,534줄이 유사 라인으로 탐지되었다. 그 결과 A 기준 유사도는 69.8%인 반면, B 기준 유사도는 30.1%로 나타났다. 이는 A03의 관련 구현이 B의 여러 모듈에 분산된 구조와 관련된다.

5.2 문자열 정성 분석 결과

문자열 비교에서는 기능 수행에 필수적이지 않은 내부 공백까지 동일하게 나타났다. A와 B의 LoginForm.Designer.cs에서는 언어 선택 라벨의 문자열을 다음과 같이 동일하게 설정하고 있었다. 특히 '언'과 '어' 사이에 삽입된 공백 5개의 위치와 개수가 모두 일치하였다.

```
this.labelLanguage.Text = "언     어";
```

이러한 일치는 단순히 두 프로그램이 동일한 언어 선택 기능을 제공한다는 사실과 구분된다. 화면에 '언어'를 표시하는 방법은 여러 형태로 구현할 수 있으며, 공백의 개수는 기능상 정해진 값이 아니기 때문이다. 따라서 문자열의 내용과 임의적인 공백 배치가 함께 일치한 점은 두 프로그램의 작성 방식에 공통점이 있음을 보여준다.

문자열 비교 결과는 정량 유사도의 해석을 보완하는 구체적인 근거가 된다. 문자열 리터럴은 이진파일 내부에 고정된 값으로 저장되므로 제어 구조보다 역컴파일 과정의 재구성 영향을 적게

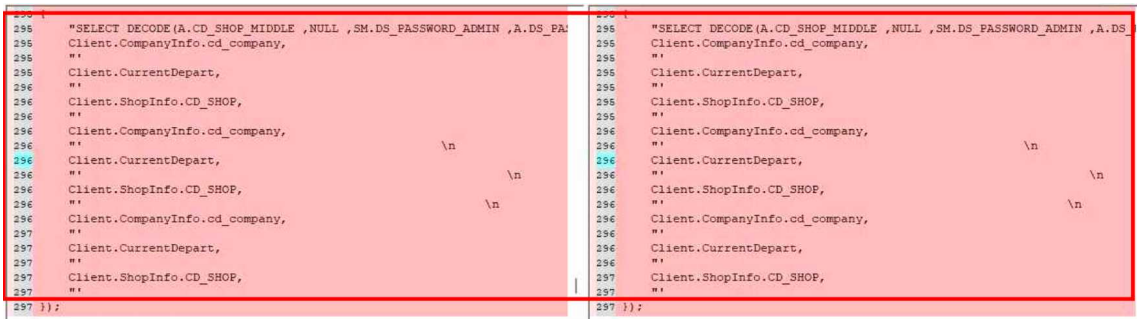

그림 2. SQL 질의문 구성 사례
Fig. 2. Example of SQL Query Construction

받는다. 따라서 기능과 관계없는 공백 배치까지 일치한 사실은 정량 분석에서 탐지된 유사 구간이 실제 구현상의 공통점과 연결됨을 보여준다.

5.3 SQL 질의문 정성 분석 결과

SQL 질의문 비교에서는 데이터 참조 요소와 문자열 구성 방식이 두 프로그램에서 동일하게 나타났다. A와 B의 MainForm.cs에서는 동일한 테이블과 열을 참조하는 장문의 질의문이 확인되었으며, 문자열을 연결하는 순서도 일치하였다. 해당 질의문 구성 사례는 그림 2와 같다.

그림 2에서 확인할 수 있듯이, 두 질의문은 내용뿐만 아니라, 줄바꿈문자와 공백의 배치도 동일하다. SQL에서는 여러 공백과 줄바꿈문자가 질의문의 실행 결과에 영향을 주지 않으므로, 개발자는 동일한 질의문을 다양한 형태로 작성할 수 있다. 그럼에도 줄바꿈문자 앞의 공백 개수와 개행 위치까지 일치한 점은 기능상 요구되는 공통점과 구분되는 구현 특징이다.

SQL 질의문 비교 결과는 두 프로그램의 데이터 처리 방식이 코드 수준에서도 공통됨을 보여준다. 구체적인 테이블과 열, 조건절, 문자열 연결 순서 및 공백·개행 배치가 함께 일치하였기 때문이다. 이 결과는 정량 분석에서 탐지된 유사 구간이 실제 데이터 접근 구조와 질의문 작성 방식의 공통점을 포함하고 있음을 뒷받침한다.

5.4 제어 구조 정성 분석 결과

제어 구조 비교에서는 조건식의 내용과 검사 순서 및 중첩 방식이 두 프로그램에서 유사하게 나타났다. A와 B의 복원 코드에서는 여러 조건을 순차적으로 검사한 뒤 특정 조건에서 다시 하위 조건을 확인하는 중첩 분기 구조가 공통으로 확인되었다. 두 프로그램에서 확인된 실제 제어 구조의 비교 사례는 그림 3과 같다.

```
if (!(a == "KOREA"))
{
    if (!(a == "ENGLISH"))
    {
        if (!(a == "CHINA"))
        {
            if (!(a == "JAPAN"))
            {
                if (a == "VIETNAM")
                {
                    Client.Language = Client.UsLanguage.VIETNAM;
                }
            }
            else
            {
                Client.Language = Client.UsLanguage.JAPAN;
            }
        }
        else
        {
            Client.Language = Client.UsLanguage.CHINA;
        }
    }
    else
    {
        Client.Language = Client.UsLanguage.ENGLISH;
    }
}
else
{
    Client.Language = Client.UsLanguage.KOREA;
}
```

그림 3. 중첩 조건문 비교 사례
Fig. 3. Comparison Example of Nested Conditional Statements

제어 구조의 공통점은 단순히 동일한 기능을 수행한다는 사실보다 구체적인 구현 방식을 보여준다. 하나의 처리 절차도 조건의 순서, 분기 위치 및 중첩 깊이를 다르게 구성할 수 있으나, 두 코드에서는 이러한 요소가 유사하였다. 이는 정량 분석에서 탐지된 유사 구간이 조건 처리 방식의 공통점까지 포함하고 있음을 보여준다.

6. 고 찰

정량 분석과 정성 분석은 두 프로그램의 코드 수준 공통점을 일관되게 보여주었다. 파일별 유사도와 함께 문자열의 공백 배치, SQL 질의문의 구성 및 조건 검사 순서가 공통으로 확인되었다. 이는 탐지된 유사 구간이 기능뿐 아니라 구체적인 구현 방식과도 연결되어 있음을 의미한다.

본 감정 절차는 중간언어와 메타데이터가 남는 프로그램, 즉 하이브리드 구현(hybrid implementation) 언어의 프로그램에 특히 적합하다. C#과 F#은 .NET 중간언어와 타입 정보를 포함하며, Java와 Kotlin도 JVM(java virtual machine) 바이트코드와 클래스 구조 정보를 포함한다. 따라서 언어별 역컴파일 도구와 정제 기준을 적용하면 이들 언어에도 활용할 수 있다.

네이티브 이진파일의 경우에는 타입과 메타데이터가 적게 남으므로 구조 정보를 보완하는 방법이 필요하다. 함수 경계, 호출 관계, 명령어 나열, 문자열 및 제어 흐름 그래프를 비교하고, 명령어를 중간 표현으로 정규화하여 함수나 기본 블록 단위의 대응 관계를 설정할 수 있다. 이를 통해 컴파일러와 최적화에 따른 표현 차이를 줄이고 실행 구조를 중심으로 비교할 수 있다.

NS 다이어그램은 서로 다른 문법으로 표현된 제어 구조를 비교하는 데 활용할 수 있다. 그림 4와 그림 5의 if-else와 switch는 문법은 다르지만, 동일한 조건을 판별하여 같은 처리로 분기한다. 따라서 조건과 실행 순서를 공통된 구조로 표현하면 역컴파일 결과의 구조적 공통점을 명확하게 해석할 수 있다.

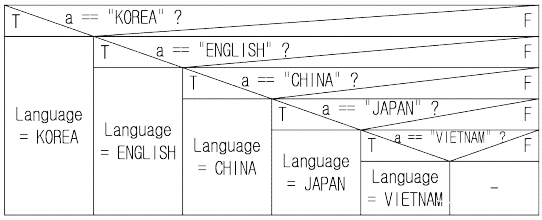

그림 4. NS 다이어그램의 분기 표현(if-else 구조)
Fig. 4. Branch Representation in an NS Diagram (if-else Structure)

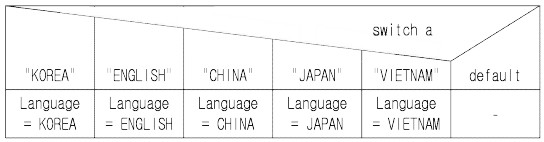

그림 5. NS 다이어그램의 분기 표현(switch 구조)
Fig. 5. Branch Representation in an NS Diagram (switch Structure)

7. 결 론

본 연구는 역컴파일 기법을 이용하여 C# 이진파일을 감정할 수 있음을 보였다. 실제로 실무 수준의 C# 프로그램 이진파일 쌍을 대상으로, 소스코드를 복원하고, 분석 대상 정제와 대응 파일 선정을 거쳐 정량 분석과 정성 분석을 수행하였다. 그 결과 전체 유사도는 원본 기준 80.1%(비교본 기준 79.0%)로 산출되었으며, 문자열 리터럴(특히 SQL 질의문) 및 제어 구조의 유사성도 확인할 수 있었다.

본 연구의 결과는 역컴파일로 복원된 코드에서도 수치와 구현 특징을 함께 이용하여 프로그램 사이의 공통점을 해석할 수 있음을 보여준다. 특히 외부 라이브러리와 자동 생성 코드를 제외

하고 파일 구성의 차이를 반영하여 비교 범위를 설정함으로써, 이진파일만 제공된 상황에서도 코드 수준의 감정을 수행할 수 있었다. 이는 정량 유사도만 제시하는 방법을 보완하고 감정 결과의 구체적인 근거를 제시한다는 데 의미가 있다.

향후에는 다양한 언어와 실행 환경의 이진파일로 감정 범위를 확대할 예정이다. 하이브리드 구현 언어(F#, Java 및 Kotlin 등) 프로그램에는 언어별 역컴파일 도구와 분석 대상 정제 기준을 적용하여 본 감정 절차의 적용 가능성을 검토할 것이다. 네이티브 이진파일의 경우에는 함수·호출 관계와 제어 흐름을 추출하고 중간 표현으로 정규화하는 방법을 적용하여 구조적 비교 방안을 연구할 것이다.

이 과제는 부산대학교
기본연구지원사업(2년)에 의하여 연구되었음

참 고 문 헌

[1] Korea Copyright Commission, “Appraisal”, 2026, Korea Copyright Commission, https://www.copyright.or.kr/eng/service/appraisal.do

[2] T. Kamiya, S. Kusumoto, K. Inoue, “CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code”, IEEE Transactions on Software Engineering, 28(7), pp.654-670, 2002, DOI : 10.1109/TSE.2002.1019480

[3] I. D. Baxter, A. Yahin, L. Moura, M. Sant’Anna, L. Bier, “Clone Detection Using Abstract Syntax Trees”, Proceedings of the International Conference on Software Maintenance, pp.368-377, IEEE Computer Society, 1998,
DOI : 10.1109/ICSM.1998.738528

[4] J. Krinke, “Identifying Similar Code with Program Dependence Graphs”, Proceedings of the 8th Working Conference on Reverse Engineering, pp.301-309, IEEE Computer Society, 2001,
DOI : 10.1109/WCRE.2001.957835

[5] M. Tufano, C. Watson, G. Bavota, M. Di Penta, M. White, D. Poshyvanyk, “Deep Learning Similarities from Different Representations of Source Code”, Proceedings of the 15th International Conference on Mining Software Repositories, pp.542-553, Association for Computing Machinery, 2018,
DOI : 10.1145/3196398.3196431

[6] ECMA International, “ECMA-335: Common Language Infrastructure (CLI), 6th Edition”, 2012, ECMA International, https://ecma-international.org/publications-and-standards/standards/ecma-335/

[7] Z. Liu, S. Wang, “How Far We Have Come: Testing Decompilation Correctness of C Decompilers”, Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, pp.475-487, Association for Computing Machinery, 2020,
DOI : 10.1145/3395363.3397370

[8] Y. Wu, S. Feng, D. Zou, H. Jin, “Detecting Semantic Code Clones by Building AST-based Markov Chains Model”, Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering, Article 34, pp.1-13, Association for Computing Machinery, 2022, DOI : 10.1145/3551349.3560426

[9] S. Dou, Y. Wu, H. Jia, Y. Zhou, Y. Liu, Y. Liu, “CC2Vec: Combining Typed Tokens with Contrastive Learning for Effective Code Clone Detection”, Proceedings of the ACM on Software Engineering, 1(FSE), Article 70, pp.1564-1584, 2024, DOI : 10.1145/3660777

[10] S. Yang, Z. Xu, Y. Xiao, Z. Lang, W.

Tang, Y. Liu, Z. Shi, H. Li, L. Sun, “Towards Practical Binary Code Similarity Detection: Vulnerability Verification via Patch Semantic Analysis”, ACM Transactions on Software Engineering and Methodology, 32(6), Article 158, pp.1-29, 2023, DOI : 10.1145/3604608

[11] S. Yang, C. Dong, Y. Xiao, Y. Cheng, Z. Shi, Z. Li, L. Sun, “Asteria-Pro: Enhancing Deep Learning-based Binary Code Similarity Detection by Incorporating Domain Knowledge”, ACM Transactions on Software Engineering and Methodology, 33(1), Article 1, pp.1-40, 2024, DOI : 10.1145/3604611

[12] P. Shirani, S. Bhatt, A. Hailane, G.-V. Jourdan, “Towards Cross-Architecture Binary Code Vulnerability Detection”, Proceedings of the 33rd Annual International Conference on Computer Science and Software Engineering, pp.191-196, IBM Corp., 2023, DOI : 10.5555/3615924.3615947

[13] B. A. Ekanem, J. Meye, “Application of Reverse Engineering Technique in Software Forensic Analysis to Detect Infringements”, Proceedings of the World Congress on Engineering 2021, pp.191-194, Newswood Limited, 2021, https://www.iaeng.org/publication/WCE2021/WCE2021_pp191-194.pdf

[14] Horsicq, “Detect It Easy”, 2024, GitHub, https://github.com/horsicq/Detect-It-Easy

[15] dnSpy, “dnSpy v6.1.8: .NET Debugger and Assembly Editor”, 2020, GitHub, https://github.com/dnSpy/dnSpy/releases/tag/v6.1.8

[16] S. Choi, K.-G. Doh, “Assessment of exEyes’ Recall using Bellon Reference Corpus as a Benchmark”, Journal of Software Forensics, 11(1), pp.31-39, 2015, https://journal.kci.go.kr/ksavs/archive/articleView?artiId=ART002707544

저 자 소 개

천준석(Junseok Cheon)

2011.8 동서대학교 컴퓨터공학과 졸업
2013.8 부산대학교 전자전기컴퓨터공학과 석사
2013.9-현재: 부산대학교 정보융합공학과 박사과정
<주관심분야> 프로그래밍 언어, 프로그램 분석, 생성형 인공지능, 소프트웨어 시각화

김영훈(Yeonghun Kim)

2013.2 영산대학교 사이버보안학과 졸업
2024.8 부산대학교 정보융합공학과 석사
2025.3-현재: 부산대학교 정보융합공학과 박사과정
<주관심분야> 프로그래밍 언어, 웹 프로그래밍

우 균(Gyun Woo)

1991.2 한국과학기술원 전산학과 졸업
1993.2 한국과학기술원 전산학과 석사
2000.2 한국과학기술원 전자전산학과 박사
2000.3-2004.2: 동아대학교 컴퓨터공학과 교수
2004.3-현재: 부산대학교 정보컴퓨터공학부 교수
<주관심분야> 프로그래밍언어, 컴파일러, 함수형언어, 프로그램분석, 프로그램시각화, 프로그래밍교육