한국항해항만학회지 제36권 제5호(Vol. 36, No. 5), pp. 363~370, 2012 (ISSN-1598-5725/eISSN-2093-8470)
Journal of Navigation and Port Research
DOI : http://dx.doi.org/10.5394/KINPR.2012.36.5.363

소프트웨어 품질 향상을 위한 리팩토링 효과 분석 : AIS 중개 서버 프로그램을 대상으로

† 이서정 · 이재욱* · 황훈규* · 이장세**

† , ** 한국해양대학교 IT공학부
* 한국해양대학교 대학원 컴퓨터공학과

Refactoring Effectiveness Analysis for Software Quality Enhancement : using AIS Mediation Server Program

† *Seo-Jeong Lee · Jae-Wook Lee* · Hoon-Kyu Hwang* · Jang-Se Lee***

† ,** *Division of IT Engineering, National Korea Maritime University, Busan 606-791, Korea*
* *Graduate School of National Korea Maritime University, Busan 606-791, Korea*

요 약 : 최근 국제해사기구에서 추진하고 있는 e-navigation은 선박의 안전운항을 위한 다양한 서비스를 지향하고 있다. 이에 따라, 해양 분야에 다양한 소프트웨어 개발이 기대되고 있으며, 동시에 그 품질의 중요성이 높아지고 있다. 이 논문에서는 소프트웨어의 품질 향상의 실험을 위해, 기존 AIS중개서버 프로그램에 소프트웨어 리팩토링(refactoring) 기법을 적용하였으며, 그 효과를 분석하였다. 리팩토링은 소프트웨어를 소스 코드 수준에서 구조의 복잡성을 줄여서, 이해하기 쉽고 기능 변경이 용이한 상태로 변경하는 기법이다. 이를 통해, 겉으로 보이는 동작의 변화는 없이 내부구조가 변경된다. 리팩토링 적용 효과 분석을 위해서, IEC/ISO 9126 소프트웨어 품질표준의 유지보수성과 관련된 기존 메트릭의 산술적 측정기법을 도입했다.

핵심용어 : AIS 중개 서버 프로그램, 소프트웨어 품질, 리팩토링, IEC/ISO 9126, 유지보수

Abstract : *Recently, International Maritime Organization has been developing e-navigation implementation strategy plan, which is focused on various services for vessel safety navigation. Then, different kinds of software will be developed in maritime area and ,with this, the quality issues are to be expected becoming more important. In this paper, we adopt software refactoring techniques to reduce the complexity of structure on source code level. It makes software program more effective to understand and modify, without any change of outward behavior. The existing AIS broadcast server program is used as an example for our trial, and calculating coupling and cohesion metric are introduced to analyze the refactoring effect, taking account of the maintainability of IEC/ISO9126 software quality standards.*

Key words : *AIS Mediation Server Program, Software Quality, Refactoring, IEC/ISO 9126, Maintenance*

1. 서 론

범용 소프트웨어 품질 향상의 필요성은 계속해서 강조되어 왔으며, 상대적으로 잘 고려되어지고 있다. 하지만, 해양 분야에 사용되는 소프트웨어의 품질 향상에 관한 내용은 상대적으로 크게 고려되지 않았으나, 최근 IMO NAV 위원회에서 제안되는 등 그 중요성이 점차 높아지고 있다(IMO 2011, IALA 2011).

일반적으로 소프트웨어는 개발된 후, 사용자가 사용하는 동안에도 지속적으로 요구사항이 변화할 수 있기 때문에 이를 위해 유지보수가 필요하다. 이 때, 변경된 요구사항을 만족하도록 단순히 소스 코드의 특정 부분만을 수정한다면, 소스 코드는 설계의 본연적인 의도와는 다르게 커지거나 복잡하게 되어 효율성을 저하시킬 우려가 있다. 또한 향후 또 다른 요구사항이 변화되거나, 개발자가 변경되었을 경우에는 쉽게 수정할 수 없거나 코드를 쉽게 이해할 수 없게 되는 문제점이 발생한다(권, 2009; 이 등, 2001).

이러한 문제점을 해결하기 위하여 리팩토링(refactoring) 기법이 제안되었다. 리팩토링은 결과의 변경 없이 소스 코드의 구조를 재조정하여 이해하기 쉽게 변경하는 것을 말한다. 즉, 소프트웨어를 개발하고, 요구사항이 변경되어 유지보수 할 때에도 항상 리팩토링을 고려하여 소스 코드를 작성하는 것이다. 리팩토링을 고려하여 소프트웨어를 개발하면, 향후 요구사항이 변화되거나, 개발자가 변경되었을 경우에도 코드를 쉽게 이해하고 수정할 수 있게 도움을 준다. 즉, 리팩토링은 소프트웨어 품질을 향상시키기 위한 기법 중 하나이다.

이 논문에서는 해양 분야와 관련된 소프트웨어 품질을 향상

† 주저자·교신저자 : 연회원, sjlee@hhu.ac.kr 051)410-4578
* 연회원, hungyu@hhu.ac.kr, wookis6766@naver.com 051)410-5227/4888
** 연회원, jslee@hhu.ac.kr 051)410-4577

시키기 위하여 리팩토링 기법을 적용하고, 그 효과를 분석하는 것에 관한 내용을 다룬다. 리팩토링 기법을 적용할 대상으로는 AIS 중개 서버 프로그램(이 등, 2011)을 사용한다. 이 논문의 구성은 다음과 같다. 2장에서 소프트웨어의 품질과 리팩토링 기법에 관한 내용을 기술하고, 3장에서 AIS 중개 서버 프로그램에 리팩토링 기법을 적용하는 것에 관한 내용을 다루며, 4장에서 리팩토링의 적용 결과를 보여주고 결론으로 끝을 맺는다.

2. 관련 연구

2.1 리팩토링의 정의 및 기법

리팩토링(refactoring)이란, 소프트웨어의 내부 구조를 보다 쉽게 이해할 수 있고, 적은 비용으로 유지보수 및 수정할 수 있도록 하기 위하여 겉으로 보이는 동작의 변화 없이 내부 구조를 변경하는 작업을 말한다. 즉, 리팩토링은 소프트웨어에서 외부에 나타나는 동작을 유지하면서 소프트웨어의 내부 구조를 개선하는 것이다(이 등, 2001; Fowler, 2002).

소프트웨어에 관한 유지보수 등의 작업을 어렵게 하는 네 가지 이유는 다음과 같다. 첫째, 읽기 어려운 소스 코드를 가진 소프트웨어는 수정하기 어렵다. 둘째, 소스 코드에 중복된 로직을 가지고 있는 소프트웨어는 수정하기 어렵다. 셋째, 실행 중인 소프트웨어의 소스 코드를 변경해야 하는 경우는 수정하기 어렵다. 넷째, 소스 코드에 복잡한 조건문이 포함된 소프트웨어는 수정하기 어렵다. 이와 같은 문제를 해결하기 위해 소프트웨어를 리팩토링하는 작업이 필요하며, 소프트웨어의 리팩토링에서는 다음과 같은 네 가지 원칙이 강조된다. 첫째, 소스 코드는 읽기 쉬워야한다. 둘째, 모든 로직이 중복되지 않도록 한 곳에서만 존재해야한다. 셋째, 기존 동작을 위험에 빠뜨릴 수정을 허용하지 않아야한다. 넷째, 조건문은 가능한 단순하게 표현해야한다(Fowler, 2002).

대표적으로, 마틴 파울러는 소프트웨어 소스 코드 내에 나타나는 무질서(Bad smell)를 정의하고, 이러한 무질서를 제거하기 위한 여러 리팩토링 기법을 제안하였다. Table 1에서 소스 코드 내에 나타나는 무질서의 종류와 그것을 해결하기 위해 사용되는 리팩토링 기법을 나열하였다(Fowler, 2002).

Table 1 Refactoring methods for bed smells

무질서	리팩토링 기법
중복된 코드 (Duplicated Code)	Extract Method Pull Up Method Form Template Algorithm Substitute Algorithm
긴 메소드 (Long Method)	Extract Method Replace Temp with Query Introduce Parameter Object Preserve Whole Object Replace Method with Method Object Decompose Conditional
거대한 클래스 (Large Class)	Extract Class Extract Subclass Extract Interface Duplicate Observed Data
긴 파라미터 리스트 (Long Parameter List)	Replace Parameter with Method Preserve Whole Object Introduce Parameter Object
확산적 변경 (Divergent Change)	Extract Class
산탄총 수술 (Shotgun Surgery)	Move Method Move Field Inline Class
기능에 대한 욕심 (Feature Envy)	Move Method Extract Method
데이터 덩어리 (Data Clumps)	Extract Class Introduce Parameter Object Preserve Whole Object
기본 타입에 대한 강박관념 (Primitive Obsession)	Replace Data Value with Object Replace Type Code with Class Replace Type Code with Subclasses Replace Type Code with State/Strategy Extract Class Introduce Parameter Object Replace Array with Object
Switch 문 (Switch Statements)	Extract Method Move Method Replace Type Code with Subclasses Replace Type Code with State/Strategy Replace Conditional with Polymorphism Replace Parameter with Explicit Methods Introduce Null Object
평행 상속 구조 (Parallel Inheritance Hierarchies)	Move Method Move Field
게으른 클래스 (Lazy Class)	Collapse Hierarchy Inline Class
추측성 일반화 (Speculative Generality)	Collapse Hierarchy Inline Class Remove Parameter Rename Method
임시 필드 (Temporary Field)	Extract Class Introduce Null Object
메시지 체인 (Message Chains)	Hide Delegate Extract Method Move Method
미들 맨 (Middle Man)	Remove Middle Man Inline Method Replace Delegation with Inheritance
부적절한 친밀 (Inappropriate Intimacy)	Move Method Move Field Change Bidirectional Association to Unidirectional Extract Class Hide Delegate Replace Delegation with Inheritance
다른 인터페이스를 가지는 대체 클래스 (Alternative Classes with Different Interfaces)	Rename Method Move Method Extract Superclass
불완전한 라이브러리 클래스 (Incomplete Library Class)	Move Method Introduce Foreign Method Introduce Local Extension
데이터 클래스 (Data Class)	Encapsulate Field Encapsulate Collection Remove Setting Method Move Method Extract Method Hide Method
거부된 유산 (Refused Bequest)	Push Down Method Push Down Field Replace Inheritance with Delegation
주석 (Comments)	Extract Method Rename Method Introduce Assertion

2.2 리팩토링과 소프트웨어 품질

국제적 소프트웨어 품질 평가 표준 ISO/IEC 9126(ISO/IEC, 2001)은 총 4개의 파트로 구성되어 있다. 파트 1은 소프트웨어 품질 평가의 전반적인 부분, 파트 2는 외부 품질, 파트 3은 내부 품질, 파트 4는 사용에 관한 품질에 관해 다루고 있다. 기능성(Functionality), 신뢰성(Reliability), 사용성(Usability), 효율성(Efficiency), 유지보수성(Maintainability), 이식성(Portability)의 품질 모델을 기준으로 각각에 해당하는 세부적인 메트릭(metric)즉, 평가도구를 제시하여 소프트웨어의 품질을 평가하도록 정의되어 있다. Table 2에 각 품질 모델별 세부적인 메트릭을 나타내었다.

Table 2 Assessment metrics of software quality

품질 모델	메트릭
기능성 (Functionality)	적합성(Suitable metrics) 정확성(Accuracy metrics) 상호 운용성(Interoperability metrics) 보안성(Security metrics) 기능 준수성(Functionality compliance metrics)
신뢰성 (Reliability)	성숙성(Maturity metrics) 결함 허용성(Fault tolerance metrics) 복구성(Recoverability metrics) 신뢰 준수성(Reliability compliance metrics)
사용성 (Usability)	이해성(Understandability metrics) 학습성(Learnablility metrics) 운영성(Operability metrics) 선호도(Attractiveness metrics) 사용 준수성(Usability compliance metrics)
효율성 (Efficiency)	시간 반응성(Time behaviour metrics) 자원 사용성(Resource utilization metrics) 효율 준수성(Efficiency compliance metrics)
유지보수성 (Maintainability)	**분석성(Analysability metrics)** **변경성(Changeability metrics)** 안정성(Stability metrics) 시험성(Testability metrics) 유지보수 준수성(Maintainability compliance metrics)
이식성 (Portability)	적응성(Adaptability metrics) 설치성(Installability metrics) 공존성(Co-existence metrics) 대체성(Replaceability metrics) 이식 준수성(Portability compliance metrics)

3. 리팩토링 기법 적용 : AIS 중개 서버 프로그램

이 논문에서 다루는 리팩토링 대상은 HTML 5의 웹 소켓 기술을 활용해 웹 브라우저에서 AIS 선박 정보를 모니터링하는 시스템의 구성요소 중 하나인 AIS 중개 서버 프로그램이다(이 등, 2011).

AIS 선박 모니터링 시스템은 Fig. 1과 같이 데이터베이스, AIS 중개 서버 프로그램, 웹 소켓 서버, 웹 브라우저 등으로 구성된다. AIS 중개 서버 프로그램은 AIS 메시지를 수신하고 가공한 뒤, 데이터베이스 저장하고 웹 소켓 서버에 이를 알린다. 웹 소켓 서버는 새롭게 저장된 데이터베이스를 읽고, 그 정보를 웹 브라우저로 전달하며, 전달된 정보는 웹 브라우저에 표시되는 메카니즘을 갖고 있다.

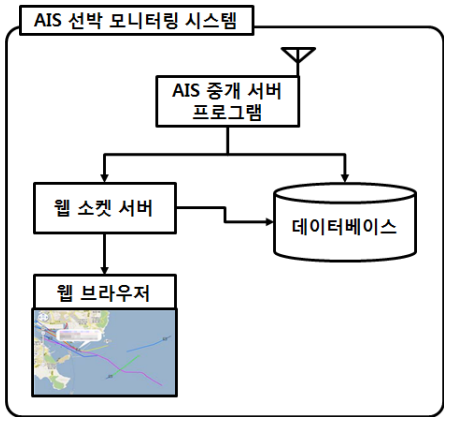


Fig. 1 Architecture of AIS ship monitoring system

3.1 AIS 중개 서버 프로그램 및 리팩토링 이슈

AIS 중개 서버 프로그램은 Fig. 2와 같이 메시지 수신, 메시지 파싱, 선박 정보 관리, 위치 정보 필터링, 데이터 전달, 데이터 저장 기능이 포함되어 있다.

- **메시지 수신** : 메시지 수신 모듈은 시리얼 통신을 통해 AIS 리시버로부터 AIS 메시지를 수신하는 기능을 한다. 데이터 전송량을 줄이기 위해 6비트로 되어 있는 AIS 메시지를 IEC 61162에 정의되어 있는 알고리즘을 이용하여 8비트의 문자로 변환해준다.
- **메시지 파싱** : 메시지 파싱 모듈은 수신된 AIS 메시지로부터 AIS 선박 정보 모니터링 시스템에 필요한 선박 정보를 파싱하는 역할을 한다. 선박 정보는 정적 정보, 항해 관련 정보, 위치 정보 등이 포함된다.

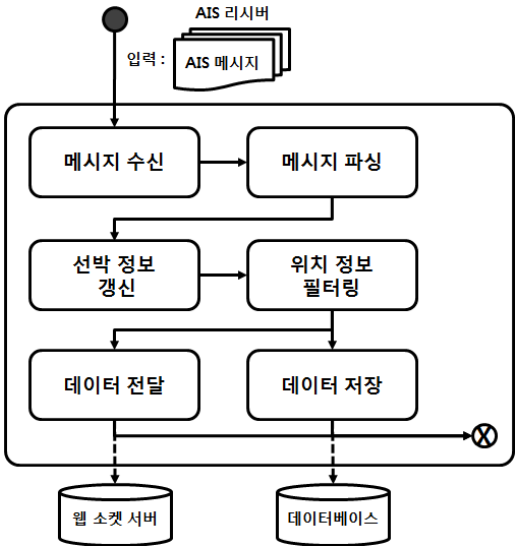


Fig. 2 Flow diagram of AIS mediation server program

- **선박 정보 관리** : 데이터베이스 접근 시간을 줄이기 위해 선박 정보는 일정 시간동안 메모리에 저장된다. 선박 정보 관리 모듈은 새로운 AIS 메시지가 수신될 때마다 메모리에 저장된 기존의 선박 정보를 갱신시키는 등의 관리 역할을 한다.

- **위치 정보 필터링** : 선박 상태에 큰 변화가 없는 경우, 중복되거나 유사한 AIS 메시지가 반복적으로 수신될 수 있다. 이러한 정보를 모두 데이터베이스에 저장하는 것은 비효율적이기 때문에, 위치 정보 필터링 모듈은 중복되거나 유사한 정보를 제거하는 역할을 한다.
- **데이터 전달** : 데이터 전달 모듈은 필터링된 데이터를 웹 소켓 서버로 전달하는 역할을 한다. 웹 소켓 서버로 전달된 데이터는 웹 브라우저를 통해 모니터링하기 위한 목적으로 사용된다.
- **데이터 저장** : 데이터 저장 모듈은 필터링된 데이터를 메시지 타입(AIS 메시지 번호)에 따라 구분하여 데이터베이스 테이블에 저장하는 역할을 한다.

리팩토링 대상인 AIS 중개 서버 프로그램은 AIS 선박 모니터링 시스템에서 메시지를 처리한 다음, 처리된 데이터를 전달하고 저장하는 핵심적인 역할을 가지고 있다. 이 프로그램은 현재 AIS장비에서 수신되는 데이터만 다루고 있지만, 향후 선박의 안전운항 및 관련 서비스를 제공하기 위해 다양한 데이터를 다룰 수 있도록 확장 계획에 있다. 또한, HTML5라는 새로운 웹브라우징 기술을 적용하고 있지만, 다른 플랫폼과 호환에 대한 요구가 발생할 수 있다. 이러한 확장에 대한 요구 및 시스템의 유지보수에 효율적으로 대비하기 위해서 프로그램 내부의 체계적 정비가 필요한 상황이다. 이 논문에서는 동작은 변경하지 않고 내부 구조를 변경하는 리팩토링 작업을 통해 이를 대비하고자 한다. 리팩토링을 위해서는 먼저, 소프트웨어의 구조 및 소스 코드의 무질서를 찾아야하며, 이러한 무질서를 없애기 위하여 적절한 리팩토링 기법을 적용해 본다.

3.2 리팩토링 기법의 적용

AIS 중개 서버 프로그램을 리팩토링하기 위해 분석해보면, Fig. 3과 같이 *MainForm* 클래스에 모든 기능이 구현되어 있어 클래스 및 메소드가 거대하다. 또한 많은 분기문(switch-case) 및 중복된 코드로 인한 무질서가 포함되어 있다. 이러한 소스 코드의 무질서를 해결하기 위하여 다음과 같은 순서로 리팩토링 기법을 적용했다.

(1) AIS 중개 서버 프로그램에서 모듈간의 AIS 메시지 데이터 교환 형식을 정의하고 있는 부분을 **Extract Class** 및 **Extract Method** 기법을 적용하여 *Message* 클래스로 추출한다. *Message* 클래스는 AIS 메시지 및 관련 정보를 포함한다.

(2) *Message* 클래스는 포함하고 있는 AIS 메시지의 형식에 따라 다른 모듈은 각기 다른 동작을 한다. 그렇기 때문에 *Message* 클래스는 각 AIS 메시지 형식 별로 **Replace Type Code with Subclasses** 기법을 적용하여 상속구조를 결정하고, **Replace Conditional with Polymorphism** 기법을 적용하여 AIS 메시지의 형식에 따른 Message 클래스를 상속한다.

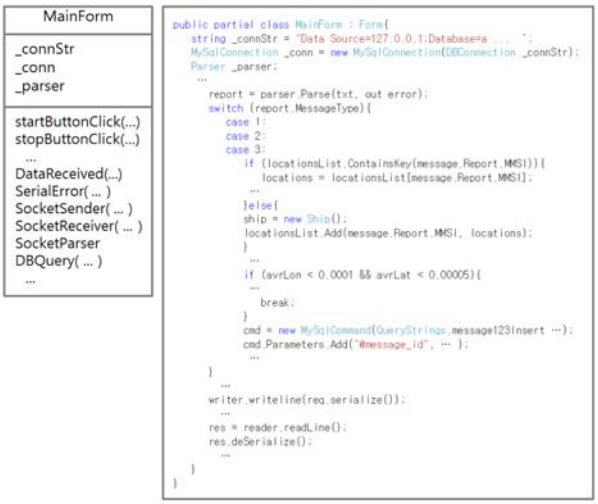


Fig. 3 Class and main source code of AIS mediation server program

(3) 메시지 수신 및 메시지 파싱 기능을 하는 부분을 **Extract Class** 및 **Extract Method** 기법을 적용하여 클래스 및 메소드를 *AISSerialPort* 클래스로 추출한다.

(4) 선박 정보 관리 기능을 하는 부분을 **Extract Class** 및 **Extract Method** 기법을 적용하여 *ShipManagement* 클래스로 추출한다.

(5) 위치 정보 필터링을 하는 부분은 인터페이스를 통해 *Message* 클래스가 포함하고 있는 AIS 메시지의 형식에 따라 상속하여 동작한다. 그렇기 때문에 **Extract Method, Move Method, Rename Method** 기법을 적용하여 인터페이스 클래스인 *ILocationFilter*의 메소드로 재정의(overriding)를 통해 동작하도록 만든다.

(6) 데이터 전달 기능을 하는 부분을 **Extract Class** 및 **Extract Method** 기법을 적용하여 *AISSocketServer* 클래스로 추출하고, AIS 중개 서버 프로그램과 웹 소켓 서버간 통신관련 코드의 효율적인 구조 개선을 위해 넌-블록킹(non-blocking) 구조로 재작성 한다.

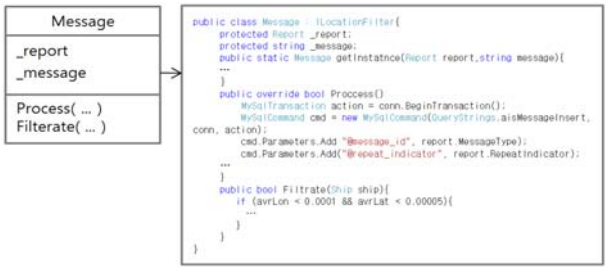
(7) 데이터 저장 기능을 하는 부분 중 **Extract Class** 및 **Extract Method** 기법을 적용하여 *DBConnction* 클래스로 추출하여 데이터베이스와의 연결을 관리한다. 데이터를 저장하고 읽는 질의문(query)은 각 *Message* 클래스에서 4형식에 따라 다르므로, 상위 *Message* 클래스에서 정의된 메소드를 각 하위 *Message* 클래스에서 재정의하여 동작하도록 만든다.

3.3 리팩토링 기법 적용 전/후의 비교

리팩토링 기법을 적용한 후, AIS 중개 서버 프로그램의 구조는 Fig. 3과 같은 형태에서 Fig. 4와 같은 형태로 변화하였다. 서로 관련 있는 클래스 및 메소드 단위로 추출, 이동, 수정되어 모듈화 된 모습을 Fig 4의 (a)~(f)에 나타냈다.

또한 AIS 중개 서버 프로그램의 흐름은 Fig. 2에서 Fig. 5와 같이 변경되었다. Fig.2와 Fig.5는 모듈화 되기 전후의 프

로그램 구조를 개념적으로 보여준다. 리팩토링 이전 프로그램은 AIS정보를 관리하고 필요한 정보를 추출하여 서버에 전

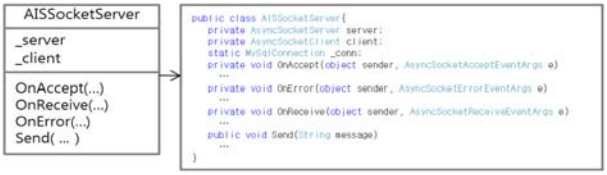


(a) 데이터 교환 형식 정의

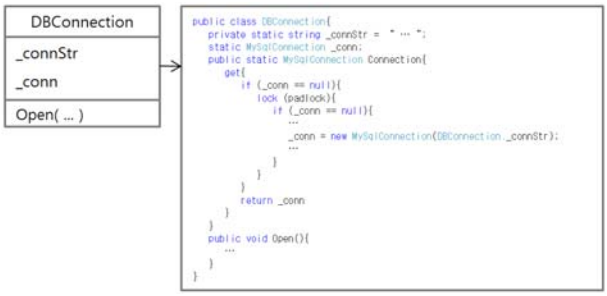
(b) 메시지 수신 및 메시지 파싱

(c) 선박 정보 관리

(d) 위치 정보 필터링



(e) 데이터 전달



(f) 데이터베이스 관리

Fig. 4 Class and main source code of AIS mediation server program after refactoring

달하는 기능이 모두 하나의 모듈에 구성이 되어 있었다. 이를 Fig.5와 같이 기능별 컴포넌트를 도입하는 방식으로 설계를 변경하였다.

즉, Fig.5의 (a)의 Message클래스와 (b)의 AISSerialPort클래스, (c)의 ShipManagement클래스와 (d)의 ILocationFilter클래스, 그리고 (e)의 AISSocketServer클래스와 (f)의 DBConnection 클래스를 각각 하나의 컴포넌트로 구성하여 컴포넌트간 독립적 역할을 수행하는 방식으로 리팩토링을 적용하였다.

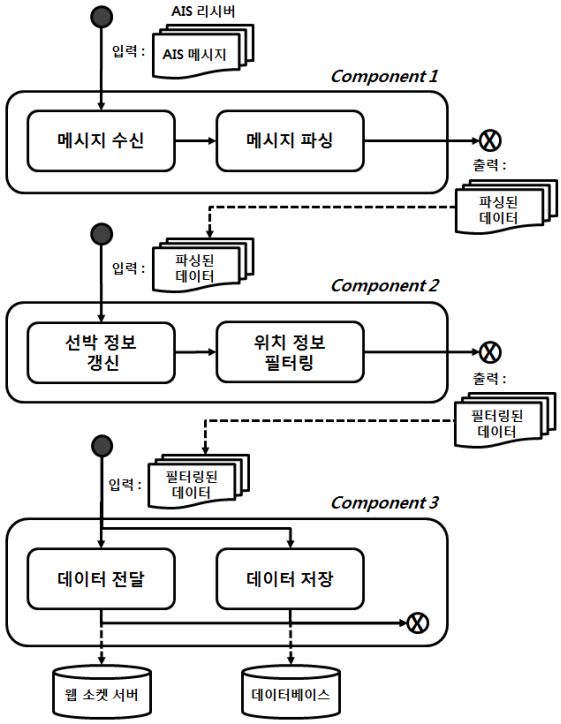


Fig. 5 Flow diagram of AIS mediation server program after refactoring

3.4 리팩토링 기법 적용 결과 분석

리팩토링의 결과를 산술적으로 계산하기 위해 다양한 연구가 제안되었다. Table 3은 리팩토링을 적용하고 소프트웨어 품질을 평가하기 위해 제안된 방법들이다.

리팩토링 결과를 분석하기 위하여 본 논문에서 다루는 프로그램 및 적용 목적이 비슷한 (최·이, 2006)가 제안한 방법을 택하여, 효과를 수치화하였다. 이 방법은 컴포넌트 기반 시스템에 적용 가능한 컴포넌트 매트릭스를 통해 각 컴포넌트 응집도(CCH)와 컴포넌트 결합도(CCP)를 계산하였다. 컴포넌트 응집도는 클래스 연관도(CC), 즉, 컴포넌트 내의 클래스들이 서로 긴밀하게 연관되어 있는 정도를 수치화 한 것이고, 컴포넌트 결합도는 컴포넌트 간의 의존도를 수치화 한 것이다.

본 논문에서 적용한 리팩토링 결과의 분석을 위해서 먼저, 클래스 연관도를 계산한 후 각 컴포넌트의 클래스 연관도의 합에 컴포넌트의 클래스 중 2개를 취할 확률과 클래스 간 메소드 호출의 최대 가중치의 곱(MAX(W))을 나누어 각 컴포넌트의 컴포넌트 응집도를 계산하였다.

Table 3 Refactoring Effect Calculating Methods

참고문헌	대상	메트릭	리팩토링기법
Sahraoui et al. (2000)	C++ 프로그램	상속성 결합도	ExtractSuperclass, Extract Subclasses, Extract Aggregate Classes
Stroulia and Kapoor(2001)	학술용	크기 결합도	Extract Superclass, Extract abstract class
Kataoka et al. (2002)	C++ 프로그램	결합도	Extract Method and Extract Class
Bois and Mens (2003)	소규모 데모	메소드수	Extract Method, Encapsulate Field, and Pull-up Method,
Leitch and Stroulia (2003)	학술용 및 상업용	소스크기 프로시져수	Extract Method, and Move Method
Tahvildari and Kontogiannis (2003)	오픈소스	복잡도 응집도 상속성	Code Transformations
Ratzinger et al.(2005)	오픈소스	결합도	Change smells
최·이(2006)	오픈소스 프로젝트	응집도 결합도	Extract Class, ExtractSubclass, Move Class, Extract Method, Replace Temp with Query

또한, 컴포넌트 결합도는 대상 컴포넌트의 클래스에서 호출하는 다른 컴포넌트의 클래스와의 연결 강도의 총합으로 계산하였다(최, 2006). 계산 과정은 Fig. 6~Fig. 8과 같으며 계산식의 A, B, C, D, E, F는 Fig. 4의 각각의 클래스들과 대응된다.

Fig. 6은 클래스들 간 연관도에 관한 계산과정이다. 클래스 연관도는 모든 클래스를 대상으로 하며, 어떤 클래스의 메소드에서 다른 클래스의 메소드를 1회 이상 호출 할 경우 해당 메소드의 호출 유형과 호출 횟수에 따라 가중치(W)를 계산한다. 가중치는 메소드의 호출 유형에 따라 다른 가중치를 부여한다. 메소드의 호출 유형은 4가지로 생성, 삭제, 수정, 참조가 있으며 각각 0.4, 0.4, 0.15, 0.05의 가중치가 부여된다.

$$CC(A,B)=\sum_{i=1}^{1}0.4^{i}+\sum_{j=1}^{1}0.05^{j}=0.45$$
$$CC(A,C)=\sum_{i=1}^{1}0.4^{i}+\sum_{j=1}^{1}0.05^{j}=0.45$$
$$CC(C,D)=\sum_{i=1}^{1}0.05^{i}=0.05$$
$$CC(D,C)=\sum_{i=1}^{1}0.15^{i}=0.15$$
$$CC(E,C)=\sum_{i=1}^{1}0.05^{i}=0.05$$
$$CC(F,C)=\sum_{i=1}^{1}0.05^{i}=0.05$$

Fig. 6 Calculation of Class Coherency

예를 들어, 클래스 A와 B 사이에는 Open()호출의 생성관계와 DataRecieved()호출의 참조관계가 한 번씩 존재하며, 그 결합도는 CC(A,B)=0.45로 계산된다. 이러한 방식으로 Fig.4의 각 클래스간 호출 함수의 종류에 따라 결합도를 계산하였다.

- 클래스 A와 C : 생성관계 Porcess(), 참조관계 Filterate()
- 클래스 C와 D : 수정관계 Refresh(), 참조관계 Filterate()
- 클래스 E와 C : 참조관계 Send()
- 클래스 F와 C : 참조관계 Open()

Fig. 7은 컴포넌트의 응집도에 관한 계산과정이다. 컴포넌트 응집도는 컴포넌트내의 클래스 간의 호출에 따른 응집된 정도를 의미한다. 계산은 컴포넌트 내의 클래스 사이의 클래스 연관도의 총합에 컴포넌트 내에서 일어날 수 있는 클래스 간 관계의 수와 최대 가중치를 나눈다. 이 때, 최대 가중치 MAX(W)는 클래스 간의 메소드 호출에 의해 발생할 수 있는 가중치의 합인 클래스 응집도의 최대치이며 그 값은 1.563이다. 이 값은 (최·이, 2006)에서 검증한 결과이며, 어떤 두 개의 클래스 간에 생성, 삭제, 수정, 참조 관계 각각에 대한 최대 가중치를 더한 값이다.

$$CCH(Component\ \ 1)=\frac{0.45}{MAX(W)}=0.288$$
$$CCH(Component\ \ 2)=\frac{0.2}{MAX(W)}=0.158$$
$$CCH(Component\ \ 3)=\frac{0}{MAX(W)}=0$$

Fig. 7 Calculation of Component Cohesion

Fig. 8은 컴포넌트 결합도에 관한 계산과정으로, 컴포넌트 간의 상호 의존도를 정량적으로 측정하기 위한 것이다. 다른 컴포넌트에 속해 있는 클래스들과의 클래스 연관도를 이용해 계산한다.

$$CCP(Component\ \ 1)=\sum_{i=1}^{1}0.4^{i}+\sum_{j=1}^{1}0.05^{j}=0.45$$
$$CCP(Component\ \ 2)=\sum_{i=1}^{1}0.4^{i}+\sum_{j=1}^{3}0.05^{j}=0.4526$$
$$CCP(Component\ \ 3)=\sum_{j=1}^{2}0.05^{j}=0.0525$$

Fig. 8 Calculation of Component Coupling

리팩토링 이전의 프로그램은 단일 클래스(MainForm)로 동작하는 단순한 구조로서 컴포넌트 응집도는 1이고, 클래스가 하나밖에 존재하지 않기 때문에 다른 컴포넌트와의 클래스 연관도의 합으로 계산되는 컴포넌트 결합도는 계산되지 않는다. 반면, 리팩토링 후 계산 결과는 Table 4와 같이 컴포넌트의 응집도와 결합도가 상대적으로 낮아졌다.

Table 4 Component Cohesion and Coupling rates after refactoring

	Component 1	Component 2	Component 3
응집도	0.288	0.158	0
결합도	0.45	0.4526	0.0525

결과적으로 본 논문에서 대상으로 한 프로그램의 리팩토링 이전과 이후를 비교하면 리팩토링 후 컴포넌트 응집도와 컴포넌트 결합도가 상대적으로 낮아져 각 클래스 및 컴포넌트의 구조를 쉽게 파악할 수 있다. 따라서 프로그램의 유지보수성이 향상되고 각 컴포넌트 재사용이 가능해진다.

이 논문의 적용 대상은 소프트웨어 규모 측면에서 시스템이라기 보다는 브로드캐스팅 서버 프로그램으로 Table 2에서 논의하는 전체 내부품질을 다루기에는 한계가 있다. 하지만, 이러한 서브 프로그램을 통합하여 시스템이 구성되며, 리팩토링은 기본적으로 소스 코드 수준에서 시작해서 그 범위를 넓혀가는 개념을 갖고 있으므로, 내부품질 전체를 다룰 수 있는지 아닌지에 대해서는 고려하지 않아도 된다.

각 품질 기준에 대한 세부 메트릭을 표준에 근거하여 적용해본 결과, 본 실험 대상에 대해서는 유지보수성(maintainability)이 가장 적합한 효과로 판단되었다. 그 중에서도, 분석성(analysability) 메트릭과 변경성(changeability) 메트릭에 밀접한 연관이 있다. 분석성은 코드 분석의 용이함을 평가하고, 변경성은 기능의 변경을 소프트웨어에서 얼마나 쉽게 수용할 수 있는가를 평가하는 항목이다.

대상 프로그램에 리팩토링 기법을 적용한 결과, 내부 구조의 변경을 통해 소스 코드가 이해하기 쉽게 수정되었고, 그에 따라 기능의 추가, 변경 또는 삭제 작업을 기존 프로그램보다 쉽게 수 있도록 개선되었다. 즉, 내부 구조의 변경을 통해 분석성과 유지보수성에서 효과적인 개선작업이 이루어졌음을 알 수 있다.

4. 결론 및 향후 연구

해양 분야에 사용되는 소프트웨어 품질에 대한 중요성이 점차 높아지고 있는 실정이다. 이 논문에서는 소프트웨어의 품질 향상을 위하여 AIS 중개 서버 프로그램에 리팩토링 기법을 적용해보았다. 리팩토링 기법이 적용되면, 유지보수성과 관련된 소프트웨어 품질이 향상되어 유지보수나 기능 추가 시에 많은 도움을 얻을 수 있다. 또한 기존 모듈의 내부 응집력은 증가하고, 다른 모듈과의 외부 결합도는 감소하여 모듈 단위의 재사용성이 증가한다.

국내 해양관련 소프트웨어의 경우, 2014년 e-navigation 개발이후 적용될 소프트웨어 품질에 대한 요구사항을 고려하여, 향후 규모와 특성이 다른 소프트웨어를 대상으로 실험을 확대하고 다양한 메트릭을 적용해 볼 계획이다.

참 고 문 헌

[1] 권용휘(2009), “프레임워크의 관점에서 본 문제 해결을 위한 리팩토링 노하우”, 마이크로소프트웨어 2009년 3월호

[2] 이서정, 이재욱(2011), “HTML5를 활용한 온라인 지도 기반 AIS선박 모니터링 시스템 구현”, 한국항해항만학회지 제35권 제6호, pp. 463~467

[3] 이종호, 박진호, 류성열(2001), “객체지향 기반의 Refactoring 프로세스”, 정보과학회논문지 : 컴퓨팅의 실제, 제7권 제4호, pp. 299-308

[4] 최미숙, 이종석(2006), “컴포넌트 기반 시스템에서 클래스들 간의 정적 그리고 동적 특성을 적용한 컴포넌트 메트릭스”, 정보과학회논문지 : 소프트웨어 및 응용, 제33권 제7호, pp. 301~315

[5] Bois, B.D., Mens, T.(2003), “Describing the Impact of Refactoring on Internal Program Quality, Proc. of the International Workshop on Evolution of Large-scale Industrial Software Applications, Amsterdam, The Netherlands, pp.37-48

[6] Fowler M.(2002), Refactoring : Improving the Design of Existing Code, Addson-Wesley

[7] IALA(2011), “e-NAV10-7-8 Software Quality”, IALA e-NAV 기술위원회 2011 의제문서

[8] IMO(2011), “DEVELOPMENT OF AN E-NAVIGATION STRATEGY IMPLEMENTATION PLAN”, Report from the Correspondence Group on e-navigation to NAV 57, Norway

[9] ISO/IEC(2001), “ISO/IEC 9126 : Software engineering – Product quality”, ISO/IEC

[10] Kataoka, Y., Imai, T., Andou H.(2002), T. Fukaya, “A Quantitative Evaluation of Maintainability Enhancement by Refactoring”, Proc. of the International Conference on Software Maintenance, pp.576-585

[11] Leitch, R., Stroulia E.(2003), “Assessing the Maintainability Benefits of Design Restructuring Using Dependency Analysis”, Ninth International Software Metrics Symposium, pp.309-322

[12] Ratzinger, J., Fischer, M., Gall, H.(2005), “Improving Evolvability Through Refactoring”, Proc. of the 2nd International Workshop on Mining Software Repositories, pp.1-5

[13] Sahraoui, H.A., Godin, R., Miceli, T.(2000), “Can Metrics Help To Bridge The Gap Between The Improvement of OO Design Quality And its Automation?”, Proc. International Conference on

Software Maintenance, pp. 154-162

[14] Stroulia, E., Kapoor, R.V.(2001), “Metrics of Refactoring-Based Development: an Experience Report”, The seventh International Conference on Object-Oriented Information Systems, pp.113-122

[15] Tahvildari, L., Kontogiannis, K., Mylopoulos J.(2003), “Quality-Driven Software Re-Engineering”, Journal of Systems and Software, Special Issue on: Software Architecture-Engineering Quality Attributes, 66(3), pp.225-239

원고접수일 : 2011년 12월 30일
심사완료일 : 2012년 04월 25일
원고채택일 : 2012년 04월 26일