컴포넌트 기반 재사용을 위한 레거시 소프트웨어 리팩토링

조현[0] 최순규 김은영
삼성전자 소프트웨어센터
(hcho[0], soonkew, marine)@samsung.com

Refactoring Legacy Software for Component-Based Reuse

Hyun Cho[0] Soon-Kew Choi Eun-Young Kim
Software Center, Samsung Electronics

요 약

IT 기술이 급격히 변화하더라도 새롭게 개발되는 대부분 소프트웨어의 핵심 부분은 기존에 존재하는 소프트웨어를 재사용하여 구현되어지는 경우가 많다. 그러나, 소프트웨어가 최초로 개발된 후 시간이 흐르고 빈번한 수정이 가해지게되면 소프트웨어는 필연적으로 최초의 형상과 많이 달라져 소프트웨어의 효과적인 재사용을 어렵게 한다. 이러한 레거시 소프트웨어를 재사용하기 위해 Refactoring을 적용하여 레거시 소프트웨어를 컴포넌트화하고 이를 재사용하고자 한다. 또한, Refactoring에 관련된 일련의 활동을 Activity로 보고 변경 관리의 대상으로 선정하여 이를 관리함으로써 Refactoring 활동을 평가하고자 한다.

1. 서 론

삼성전자는 컴퓨터, 디지털 TV, 휴대폰 등과 같이 다양한 전자 제품을 생산하고 있다. 이러한 다양한 전자 제품들은 Digital Convergence 시대를 맞이하여 하나의 제품에서 다양한 기능을 제공할 수 있도록 통합되어 가고 있으며 과거와 같이 제품 고유 특징에 대한 경계가 모호해지고 있다. 그림 1은 삼성전자에서 생산되는 대표적인 제품과 그에 탑재되는 코어 소프트웨어를 나타내고 있다.

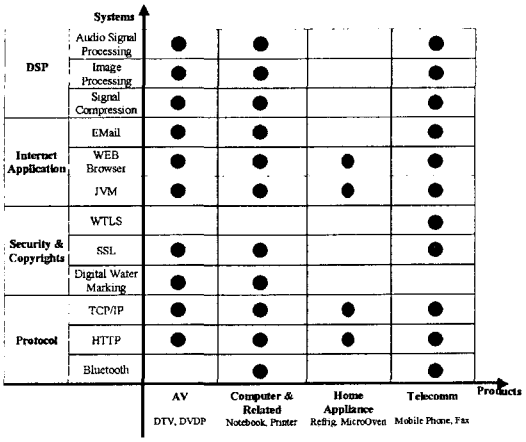

그림 1. 삼성전자 제품과 코어 소프트웨어

그림 1에서 보듯이 소비자가 원하는 기능을 제공하기 위한 디지털과 네트웍에 관련된 매우 많은 코어 소프트웨어가 존재하며 또한 새로운 기술과 기능을 접목하기 위해 더욱더 많은 코어 소프트웨어가 개발될 것이다.

지금까지 다른 새로운 기술을 제공하는 소프트웨어를 제외한 대부분의 소프트웨어는 기존 코어 소프트웨어를 재사용하고 기타 모듈을 부분적으로 수정하는 방식으로 개발이 가능할 것이다. 또한 일부 코어 소프트웨어는 다양한 제품에서 요구하는 기능을 제공함으로 재사용을 고려하여 개발하고 유지 관리하여야 한다. Web Browser나 JVM(Java Virtual machine)이 이러한 코어 소프트웨어의 좋은 예이다. 따라서, 레거시 코어 소프트웨어 재사용에 대한 요구는 개발 시간과 노력을 단축시키고 일정 수준의 품질을 확보하기 위한 전략으로 점점 더 의미가 강조되어지고 있다.

레거시 코어 소프트웨어 재사용의 한 방법으로 레거시 코어 소프트웨어를 컴포넌트로 재구성하고 재구성된 컴포넌트 중에서 재사용 가능성이 높은 컴포넌트를 대상으로 Refactoring을 적용하여 컴포넌트의 구조를 유연하고 강건하게 수정하여 재사용성을 높이고자 한다. 또한, 재사용 가능한 컴포넌트 생성을 위한 Refactoring 과정을 Activity로 정의하고 이를 변경 관리와 통합하여 관리함으로써 다음 Refactoring시의 예상되는 Activity를 정확히 예측할 수 있도록 한다.

2. Refactoring

레거시 코어 소프트웨어를 재사용 가능한 컴포넌트로 변환하기 위해 그림 2와 같은 Reengineering, Refactoring, Certification과 같이 3단계의 절차를

수행한다.

레거시 코어 시스템에서 컴포넌트를 추출하기 위해 먼저 Reengineering 과정을 통해 레거시 코어 소프트웨어의 구조와 기능에 대해 먼저 이해하고 Refactoring 정책을 설정하고 Refactoring을 실시한다. Refactoring을 통해 생성된 컴포넌트는 인증 절차를 거침으로써 완전한 컴포넌트로써 의미를 지니게 된다.

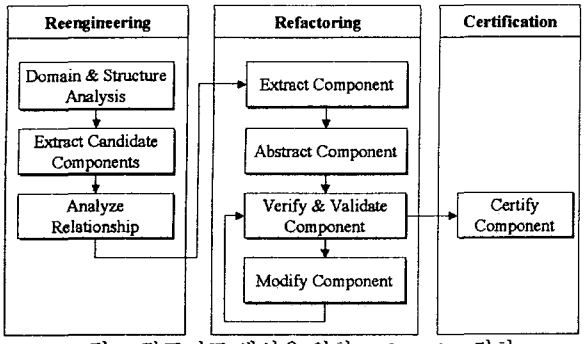

그림 2. 컴포넌트 생성을 위한 Refactoring 절차

2. 1 Reengineering 단계

Reengineering 단계에서는 레거시 소프트웨어의 Domain과 Structure를 분석하여 레거시 소프트웨어의 기능과 구조를 파악하여 후보 컴포넌트를 추출하고 그들간의 관계를 파악한다.

Domain and Structure Analysis

Domain and Structure Analysis에서는 개발 관련 문서, 코드, 개발자와의 인터뷰 등을 통하여 Refactoring하고자 하는 레거시 소프트웨어의 Architecture, Structure 및 후보 컴포넌트를 나열하기 위한 기본 정보-요구 사항 명세서, 기본 설계서, 상세설계서 등-를 수집한다.

Extract Candidate Component

Domain and Structure Analysis의 결과물인 레거시 소프트웨어의 Architecture와 Structure를 바탕으로 재사용 가능 여부를 판단하여 후보 컴포넌트를 선정한다. 재사용 컴포넌트 선정 기준으로는 하드웨어에 독립된 모듈, Call Graph 상에서 Call을 많이 받는 부분을 우선적으로 검토한다.

Analyze Relationship

선정된 후보 컴포넌트 간의 관계, 후보 컴포넌트와 레거시 소프트웨어와의 관계를 파악하여 컴포넌트의 독립성과 Dependency를 파악한다. Call Graph를 이용하면 이러한 관계를 손쉽게 파악할 수 있을 것이다. 관계 분석이 완료되면 Refactoring을 위한 사전 작업으로 Refactoring 계획서와 V&V(Verification & Validation) 계획서를 작성한다. 두 계획서 내에는 재사용 가능한 컴포넌트 추출을 위한 모든 Refatoring 활동과 컴포넌트 추출 우선 순위, 변경관리 정책 등에 대해 기술한다.

2. 2 Refactoring 단계

Refactoring 단계에서는 레거시 소프트웨어에서 실제로 컴포넌트를 추출하고 Refactoring 기법을 적용하여 컴포넌트를 생성하고 생성된 컴포넌트가 손쉽게 재사용 가능하도록 추상화하고 검증하는 과정을 거치게 된다.

Extract Component

Reengineering 단계에서 선정된 후보 컴포넌트에서 재사용 가능한 컴포넌트의 실제 코드를 레거시 소프트웨어에서 추출한다.

Abstract Component

레거시 소프트웨어에서 추출된 컴포넌트를 대상으로 추상화 작업을 실시한다. 기존 디자인 문서와 요구 사항 문서 등에서 새로이 추출된 컴포넌트에 부합하는 것을 추출하여 새로이 추출된 컴포넌트를 위한 문서화를 실시한다. 따라서, 이 단계에서 생성되는 것은 추출된 컴포넌트에 대한 아키텍쳐, 구조 설계, 상세 설계 등에 관한 문서이다. 이 단계가 종료되면 컴포넌트의 형상 관리를 위한 최초의 베이스라인을 생성한다.

Verify & Validate Component

컴포넌트 추출과 추상화가 완료되면 이에 대한 Verification과 Validation을 위한 작업을 수행한다. 즉, Refactoring 과정을 수행하는 동안 변경된 코드가 원래의 기능을 충실히 수행하는가를 확인하기 위해 필요한 Test Scenario와 Test Case를 생성하고 추출된 컴포넌트에 대한 최초의 Test를 수행하게 된다.

Modify Component

추상화 과정에서 나온 컴포넌트를 기술한 여러 문서와 실제 코드를 대상으로 이해하기 쉽고 단순한 구조를 가진 컴포넌트로 수정한다. 이 과정에서 Refactoring에 관련된 기법들이 적용되어지며 수정된 컴포넌트는 다시 V&V 과정을 거치된다. 이러한 과정을 여러 번 반복하여 수행으로써 컴포넌트에 관한 문서와 코드는 일치하게되고 재사용이 가능한 완벽한 형태의 컴포넌트로 형태를 갖추어간다.

2. 3 Certification 단계

추출된 컴포넌트를 대상으로 Refactoring를 적용하고 V&V 과정을 거치게 되면 재사용 가능한 컴포넌트가 생성되고 이를 다른 개발자와 공유하기 위하여 재사용 레포지토리에 등록한다. 재사용 레포지토리에 등록하기 위해 컴포넌트 생성에 관련된 일련의 활동과 산출물들에 대해 Certification을 진행하여 등록하려는 컴포넌트가 대한 일정한 수준의 품질을 확보하였는가를 판단한다.

3. Activity-based Change Management

레거시 소프웨어에서 재사용 가능한 컴포넌트를 생성

하기 위해 수행한 Refactoring을 포함한 모든 활동을 주기적으로 제어하고 확인하기 위하여 Activity 기반의 변경 관리 기법을 적용한다. Activity 기반의 변경 관리 기법이란 기존의 변경 관리 시스템에서 제공하는 소프트웨어에 대한 형상 및 버전 관리에 덧붙여 형상과 버전 변경에 대한 모든 활동을 Activity로 정의하고 처리 경과 및 결과에 대한 일련의 절차를 Activity에 대한 State Transition으로 관리하는 것을 의미한다. 즉, 모든 변경은 최소한 하나의 Activity와 연결되어야 하며 하나의 Activity는 조직에서 미리 정의한 State Transition에 따라 Activity의 진행 상황을 관리한다.

Reengineering 단계에서 Refactoring에 관련된 일련의 활동을 정의하기 위하여 WBS(Work Breakdown Structure)를 수행하여 Refactoring에 필요한 활동을 도출한다. WBS에 의해 나타난 모든 Activity는 Activity 기반 변경 관리 시스템에 등록되며, 등록된 Activity를 바탕으로 프로젝트 리더는 재사용 컴포넌트 개발 팀원에게 각각의 Activity를 할당하게 된다. 컴포넌트 추출 및 개선을 위한 Refactoring 활동에 앞서 모든 팀원들은 자신에게 할당된 Activity가 무엇인지를 변경 관리 시스템에서 확인하게 되며, 자신의 Activity의 상태를 Open으로 설정하게 되면, 자신에게 설정된 Activity와 컴포넌트에 관련된 Asset(코드와 문서들)이 상호 연결 고리를 가지게 되며 변경 활동이 종료되어도 이 연결 고리는 지속적으로 이어지게 된다.

즉, 문서들과 코드에 가해지는 모든 변경 활동들이 변경에 앞서 Activity로 먼저 정의되어 변경 관리 시스템에 등록되며 과제에 참여하는 모든 개발자들은 등록된 Activity를 기준으로 Activity의 Status를 변화시키며 자신에게 할당된 Activity를 처리하게 된다. 따라서, 이러한, Activity기반의 변경 관리 기법을 적용함으로써 하나의 레거시 소프트웨어를 컴포넌트 기반의 소프트웨어로 재구성할 때 필요한 Activity를 일목요연하게 파악할 수 있으며 하나의 Activity가 등록되어 처리되어 완료될 때까지 걸리는 시간을 산출할 수 있어 개발 효율과 Activity의 난이도를 산정하는 기준으로 제공된다.

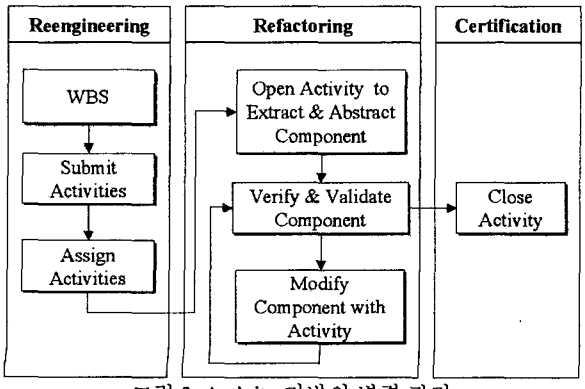

그림 3. Activity 기반의 변경 관리

위 그림 3은 Activity 기반의 변경 관리 절차와 레거시 소프트웨어에서 재사용 가능한 컴포넌트 추출을 위한 절차와 어떠한 관계를 가지는가를 보여주고 있다.

4. 결론

레거시 소프트웨어를 컴포넌트화하고 재사용 가능한 컴포넌트를 추출하기 위하여 Refactoring 적용하고 재사용 가능한 컴포넌트를 생성하는데 필요한 모든 활동을 Activity 기반의 변경 관리 기법으로 관리하여 보았다.

레거시 소프트웨어 전체를 Refactoring을 목적으로 하지 않고 레거시 소프트웨어 중에서 재사용 가능한 컴포넌트를 대상으로 Refactoring을 적용하여 상대적으로 적은 부분에 대하여 Refactoring이 적용되었지만, 기존의 레거시 소프트웨어에서 단순히 추출된 컴포넌트 보다 Refactoring이 적용된 컴포넌트는 구조적으로 안정되고 이해하기 쉬워졌으며 유연해졌다. 또한, 변경이 쉬워져 급변하는 요구 사항 변화에 적기에 대응할 수 있게 되었다.

그리고, 재사용 컴포넌트 추출에 관련된 Refactoring을 포함한 모든 활동들을 Activity 기반의 변경 관리 기법으로 관리함으로써 레거시 소프트웨어로부터 재사용 컴포넌트를 추출하는 모든 과정에 대한 시간과 노력에 대한 예측이 시스템 규모별, 개발자별로 가능해졌으며 이런 실험 데이터를 기반으로 레거시 소프트웨어에서 재사용 컴포넌트를 추출해 내는 일련의 과정을 표준 Activity로 도출하고 정의할 수 있게 되었다.

5. 참고 문헌

[1] Ivar Jacobson, Martin Griss, Patrik Jonsson, "Software Reuse", Addison-Wesley, 1997

[2] Martin Fowler, "Refactoring-Improving the Design of Existing Code", Addison-Wesley Longman Inc., 1999

[3] Hyun Cho, EunYoung Kim, "Embedded Software Development using Unified Change Management", KISS, Proceeding of the Fall Conference, 2001

[4] Rational Inc., Working with UCM