특집

CBD 도입을 위한 실천적 방법

조 남 규[1)]

목 차

1. 서 론

1990년대 중반부터 소프트웨어 산업에 적용되기 시작한 컴포넌트 기반 개발(CBD; Component-Based Development) 접근법은 독립적인 소프트웨어의 재사용을 통한 개발 비용과 시간의 단축을 강조하고 있다. 그러나 컴포넌트를 이용한 가시적인 효과는 제조나 건설 등 이미 성숙한 산업에서 우리가 경험했던 그것과 비교하였을 때 상대적으로 아직 미진한 실정이다[1]. 이러한 이유 때문에 많은 조직에서 CBD 기술 도입을 망설이고 있는 것이 사실이며, 그나마 CBD 접근법을 도입하려는 대다수의 조직에서도 기존 컴포넌트의 이용보다는 컴포넌트 개발에 초점을 맞추고 있다. 이것은 또 다른 소프트웨어의 중복 생산을 야기하게 되고 동일한 소프트웨어를 이중적으로 개발하고 관리해야 하는 문제를 발생시킨다. 컴포넌트를 획득할 수 있는 방법으로는 구매/수정(customization) 및 변경(modification)/개발 등 다양한 방법이 있지만, CBD를 조직에 도입하여 빠른 효과를 얻고자 한다면 많은 비용과 시간이 소요되는 신규 컴포넌트 개발보다는 구매나 수정을 통한 기존 컴포넌트의 재사용이 더욱 나은 접근법임이 자명하다. 또한 대부분의 조직이 소프트웨어의 생산자가 아닌 소비자라는 점을 감안한다면 기존 컴포넌트의 재사용이나 구매를 위한 구체적인 절차와 방법이 필요한 실정이다. 물론 적합한 컴포넌트를 찾지 못하였을 경우, 기존 컴포넌트의 재사용이 주어진 니즈를 만족시키기 위한 가장 경제적 방법이 될 수는 없다. 따라서 적절한 후보 컴포넌트를 식별하고 이것들에 대한 평가를 선행하여 요구사항에 만족하는 컴포넌트를 획득하기 위한 체계적인 방법이 요구된다.

최근 컴포넌트를 개발하기 위한 다양한 프로세스와 접근법이 연구되고 있지만 실질적인 컴포넌트 획득에 대한 방향은 제시되지 못하고 있다. 본 논문에서는 컴포넌트의 특성을 고려하여 조직에 맞는 후보 컴포넌트를 선별하고 컴포넌트와 요구사항 사이에 적합성을 평가하여 비즈니스 니즈에 맞는 컴포넌트를 획득하는 절차를 통해 CBD 도입을 위한 실천적 방법을 제시하고자 한다. 본 논문의 구성은 다음과 같다. 2장에서는 예비 평가를 통해 후보 컴포넌트를 선별(Short- listing)하는

1) SolutionLink 책임컨설턴트

과정에 대해서 알아본다. 3장에서는 컴포넌트와 요구사항 사이에 격차분석(Gap analysis)을 통해 후보 컴포넌트의 적합성을 평가해보고, 이러한 격차를 줄이기 위한 선택적인 방법(Solutions to gap)에 대해서 설명한다. 4장에서는 조직에서 컴포넌트 획득(acquiring)을 승인받기 위한 시안을 소개하며, 5장에서 끝을 맺도록 한다.

2. 후보 컴포넌트 식별

2.1 컴포넌트 공급자 식별

후보 컴포넌트는 조직 내부의 컴포넌트 저장소, 혹은 외부의 전문가나 시장조사를 통해서 찾을 수 있다. 때로는 컴포넌트를 찾는 것보다 패키지 공급자를 찾아보는 것이 보다 선택의 폭을 넓힐수 있는 방법이 된다.

관리적인 측면에서 본다면 우선 6~8군데의 공급자를 나열해보고, 이 중 2~4군데의 후보를 선별한 다음에 최종적으로 가장 접합한 하나의 후보에 대해서 세밀한 격차 분석(gap analysis)을 적용해보는 것이 좋다.

2.2 후보 컴포넌트 심사

처음부터 조직에 알맞은 후보 컴포넌트를 찾기보다는 우선 적절하지 못한 후보를 탈락시키는 것이 효과적인 방법이 될 수 있다. 명확한 심사 기준은 재사용 컴포넌트 목록에서 적절하지 못한 후보를 제거하는데 필수적인 요소인데, 컴포넌트를 도입하는데 있어서 다음의 심사 기준을 적용해 볼 수 있다;

- 도입하려는 소프트웨어가 조직의 목적과 요구사항에 적합한가?
- 도입하려는 소프트웨어가 기존의 다른 소프트웨어와 호환이 되는가?
- 소프트웨어 공급자로부터 충분한 지원을 받을 수 있으며, 공급자가 사라진다면 소스 코드를 사용할 수 있는가?
- 도입하려는 소프트웨어는 커스터마이징하기 쉬운가?

특히 컴포넌트를 도입하려고 한다면 이것이 조직의 컴포넌트 플랫폼상에서 올바르게 동작하는지를 확인해보는 것이 반드시 필요하다.

2.3 컴포넌트 예비 평가

소프트웨어 패키지를 획득하기 위해서는 보통 정형적 평가(formal evaluation)가 적당하지만, 컴포넌트의 경우에는 좀 더 간결한 비정형적 평가(informal evaluation)가 적절하다. 패키지를 제공하는 공급자를 선별하는 과정에서 설문조사(RFI; Request For Information)와 제안요청서(RFP; Request For Proposal)를 작성한다. 여기에 포함되어야 할 내용으로는 소프트웨어 패키지의 요구사항 충족 범위, 재정상태를 포함한 공급자 정보, 제품의 계획, 가격, 라이센스 및 주요 고객 명단, 제품 구현 정보(커스터마이징, 설치, 데이터 마이그레이션, 사용자 매뉴얼, 교육) 그리고 지원과 유지보수 정책 등이 있다.

평가를 위한 채점 방법(scoring mechanism)으로는 식별된 탈락기준에 따라 합격여부(Yes/No)만을 가지고 당락을 결정하는 방법과 각각의 평가기준마다 가중치(weight)를 부여하여 채점하는 방법이 있다. 전자의 경우에는 보통 RFI와 같은 간단한 설문조사를 위하여 사용되며, 후자는 제안서를 평가하는데 사용된다. 컴포넌트의 기능성을 평가하는데 있어서 MoSCoW 규칙을 적용하는 것 또한 효과적인 방법이 될 수 있다[2].

가중치에 의한 평가 매커니즘에서 가중치를 부여하는 방법은 다음과 같다;

- 최상위의 평가항목을 결정하고, 각 항목별로 백분율(%)을 부여한다.
- 최상위 평가항목들을 다시 각각 하위 항목로 나누고, 다시 각 하위 항목별로 백분율(%)를 부

여한다.

- 각각의 최하위 평가항목에 대해서 합격여부(Yes/No)에 따라서 0점 또는 1점을 준다.
- 기능을 현재 지원하고 있는지 아니면 미래에 지원할 예정인지에 따라서 점수를 다르게 줄 수 있으며, 기능의 수정(customization)이 필요한 경우 수정의 정도에 따라서 점수에 차등을 주는 것도 좋은 방법이다.

〈표 1〉은 가중치 평가 매커니즘에 의해서 채점한 평가표 사례를 보여주고 있다.

〈표 1〉 가중치 평가 매커니즘에 의한 평가표

No.	Criterion	Percent	Weight	Score	Weighted score
1	Interfaces	40			
2	Functionality	40			
3	Non-functional requirements	20			
1.1	...				
2.1	Order entry	40	16	1	16
2.2	Invoicing	40	16	1	16
2.3	Stock control	20	8	0	0
3.1	...				

2.4 참조 사이트 조사 및 후보 선택

제안서를 평가하는 동안에 명확하게 조사되지 못한 사항들에 대해서는 참조할만한 고객 사이트를 방문하여 핵심적인 이슈에 대한 확신을 얻도록 한다. 참조 사이트(reference site) 방문은 특히 공급자의 지원수준을 확인하고, 소프트웨어 구현의 난이도와 소프트웨어 구현에 소요되는 비용, 그리고 구현하는 동안에 발생할 가능성이 있는 문제점 등을 조사하는데 매우 유용하다.

2.5 후보 선택

예비 평가 결과를 토대로 하나의 후보 컴포넌트를 선정하여 보다 상세한 컴포넌트 적절성 평가를 수행하도록 한다. 적절성 평가에서 후보 컴포넌트가 적합하지 못한 것으로 판단되는 경우에는 다시 다음 순위의 후보 컴포넌트를 대상으로 적절성 평가를 수행한다. 모든 후보 컴포넌트를 대상으로 적절성 평가를 수행한 결과 적합한 후보 컴포넌트를 찾지 못하였을 경우 해당 컴포넌트는 일반적인 소프트웨어 개발 절차를 통해서 획득하거나, 빈번한 경우는 아니지만 요구사항을 변경하는 경우도 있다.

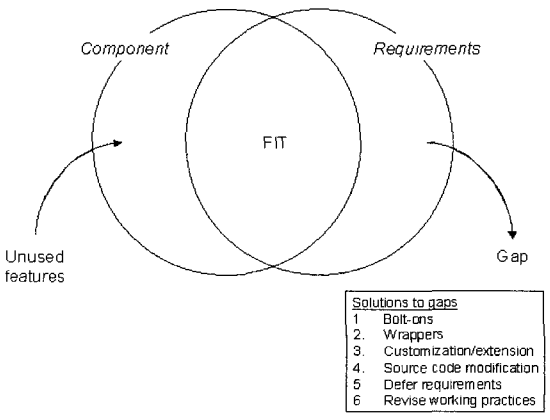
3. 적절성 평가와 변경 명세화

3.1 적절성 평가

후보 컴포넌트가 요구사항을 만족시키는지를 판단하기 위해서는 격차 분석(Gap analysis)를 수행해야 하며 격차 분석은 다음의 내용을 평가하기 위해서 사용된다;

- 후보 소프트웨어의 능력(capability).
- 후보 컴포넌트의 능력이 비즈니스 요구사항에 만족하는 정도(extent).
- 후보 컴포넌트가 요구사항에 만족할 수 있도록 컴포넌트를 수정하는데 필요한 노력(effort).
- 비즈니스 요구사항을 후보 컴포넌트에 맞도록 고칠 것인지의 여부.

(그림 1)은 컴포넌트가 제공하는 특성과 이것이 요구사항에 어느 정도 만족하는지를 보여주는 그림이다.



(그림 1) 컴포넌트 적절성 평가

우선 제3자로부터 획득한 컴포넌트인 경우에는 보통 다음의 기술적인 특성을 고려해 보아야 한다;

- 공급자가 말한 것처럼 컴포넌트가 제대로 동작하는가?
- 컴포넌트에 오류는 없는가?
- 요구되는 기능, 성능, 복구성, 또는 기타 비기능적 요구사항을 만족하는가?

다음은 컴포넌트가 제공하는 기능과 요구사항에 명세서에 기술된 기능을 매핑하여 요구되는 비즈니스 기능을 컴포넌트가 어느 정도 제공하는지 측정하여 적절한 후보를 찾도록 한다. 컴포넌트의 기능적인 특성을 평가하기 위해서는 다음을 고려해야 한다;

- 컴포넌트가 제공하는 서비스(즉, 인터페이스)와 요구사항을 어떻게 매핑할 것인가?
- 애플리케이션 컴포넌트의 경우 그것의 사용자 인터페이스와 업무 흐름이 조직의 비즈니스에 어느 정도 적합한가?
- 컴포넌트에 의해서 야기되고 소비되는 이벤트는 무엇인가?
- 컴포넌트가 관리하고 있는 데이터는 무엇인가?
- 사용자 인터페이스, 업무 흐름, 데이터 관리 등의 요구사항을 만족시키기 위해 컴포넌트를 쉽게 수정할 수 있는가?

마지막으로 기존 애플리케이션 재공학(reengineering)을 통해 컴포넌트를 획득하는 경우에 재공학의 용이성이 평가되어야 한다. 우선 필요한 로직을 호출할 수 있는 적절한 진입점(entry point)을 애플리케이션이 제공하는지 확인해 보아야 한다.

필요하지 않지만 코드의 복잡성 때문에 어쩔 수 없이 사용된 애플리케이션 로직이 존재한다면 이것을 제거할 수 있는지 살펴 보아야 한다. 만약 제거할 수 있다면 필요없는 로직을 분리하여 다시 작성하는 것이 효과적이겠지만, 이런 경우에는 소프트웨어의 재사용은 기대할 수 없다. 예를 들어, 데이터 접근 로직(data access logic)과 비즈니스 로직(business logic)이 서로 엉켜있는 상황에서 데이터 접근 로직만을 따로 추출하거나 비즈니스 로직만을 따로 추출하기 어려운 경우가 여기에 해당된다.

3.2 변경 명세화

컴포넌트를 획득하기에 앞서 컴포넌트와 요구사항 사이에 격차를 분석한 후 해당 컴포넌트의 도입이 타당하다고 판단되는 경우에는 요구사항에 대한 컴포넌트의 기능적 혹은 기술적 부족분(격차)을 해소하기 위한 방법을 선택해야 한다. 격차 분석의 결과는 보통 다음의 세가지 유형으로 나타난다.

3.2.1 기능적 불일치

소프트웨어의 기능과 요구하는 기능이 맞지 않는 경우인데, 요구사항에 기술된 기능이 없어나 업무 규칙을 처리하는 방식이 다른 경우가 이것에 해당된다. 원하는 기능 이상의 것을 제공하는 경우에도 기능적 불일치가 발생한다고 볼 수 있지만, 이것은 기능적인 측면만을 보았을 경우에는 컴포넌트 도입에 큰 영향을 끼치지 못한다.

3.2.2 데이터 불일치

업무에 필요한 데이터를 컴포넌트가 관리하지 않거나 필요없는 데이터를 관리하는 경우며, 기존 소프트웨어가 다루는 데이터와 컴포넌트가 다루는 데이터가 서로 중복되는 경우에는 기존 소프트웨어와 컴포넌트가 관리하는 데이터 사이에 동기화가 필요하다.

3.2.3 기술적 이슈

컴포넌트가 요구하는 플랫폼이 조직의 그것과 다르거나 컴포넌트가 지원하는 데이터베이스가 다른 경우, 또는 소프트웨어를 컴포넌트화

(componentize)하는데 기술적으로 많은 노력이 필요한 경우에 해당된다.
분석 결과 이러한 문제가 발생하였다면 이러한 격차를 제거하거나 줄이기 위해 컴포넌트를 수정해야 하며, 이러한 변경을 명세화(specifying changes)하여 컴포넌트 도입에 따른 위험을 최소화할 필요가 있다. 격차를 제거하기 위한 옵션은 컴포넌트를 조직내에 컴포넌트 저장소로부터 획득한 것인지 아니면 제3자로부터 구매한 것인지에 따라서 달라질 수 있으며, 소스 코드를 함께 제공되는 화이트박스(white-box) 형태의 컴포넌트인지 아니면 바이너리 형태로 제공되는 블랙박스(black-box) 컴포넌트인지에 따라서도 매우 달라진다.

3.2.4 블랙박스(black-box) 형태의 수정

컴포넌트가 파라매터를 통해 소프트웨어의 기능을 수정(customization)할 수 있는 기능을 제공하거나 컴포넌트의 수정을 위한 특별한 조립기능을 제공하는 경우에는 블랙박스 형태의 수정이 가능하다.

3.2.5 래퍼(wrapper) 또는 어댑터(adaptor)의 사용

블랙박스 형태의 재사용을 제외한다면 요구사항과 컴포넌트 사이에 격차를 줄이기 위한 가장 효과적인 방법은 래퍼 혹은 어댑터를 사용하는 것이다. 래퍼는 코드의 수정없이 기존의 소프트웨어를 컴포넌트화하여 사용하기 위한 방법이며, 어댑터는 두 개의 컴포넌트 사이에 불일치를 해결하는데 사용된다. (그림 2)는 이 두가지 옵션의 차이점을 설명하고 있다. 또한, 기존의 시스템을 래핑하기 위해서는 (그림 3)처럼 컴포넌트와 기존의 API(Application Program Interface) 사이에 데이터 혹은 기능의 불일치를 해결하기 위한 정제(refinement) 로직이 필요하게 된다.

3.2.6 요구사항의 변경

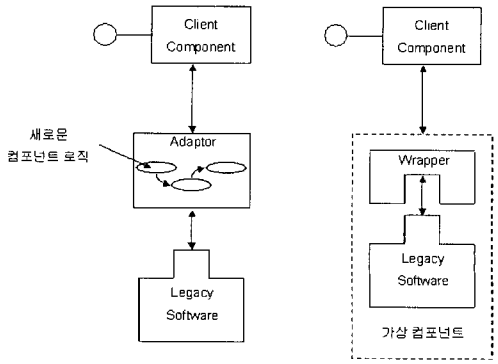
어떤 경우에는 소프트웨어의 직접적인 변경이 없이 요구사항을 변경하거나 그것의 구현을 뒤로 미룸으로써 격차를 제거할 수도 있다. 즉, 아주 특별한 경우이기는 하지만 컴포넌트를 변경하는 것이 아니라 업무 자체를 컴포넌트에 맞추는 것이 훨씬 경제적인 효과를 나타낸다.

3.2.7 소스코드(source code)의 변경

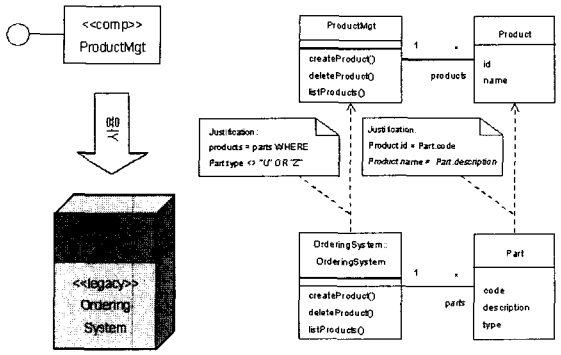
소스코드가 함께 제공되는 화이트박스 컴포넌트의 경우에는 요구사항에 만족하도록 소스코드를 직접 수정할 수도 있다. 이 경우에는 요구사항에는 가장 적합한 컴포넌트를 획득할 수 있지만 상대적으로 많은 비용이 소요되므로 가급적 기존 코드 변경의 최소화와 변경 관리가 필수적으로 요구된다. 간혹 새로운 버전의 컴포넌트를 만드는 것이 기존에 애플리케이션에서 사용되는 컴포넌트의 대체를 요구하는 경우가 있는데, 이런 경우에는 새로운 버전(new version)을 만들기 보다는 기존 컴포넌트를 복제/변경하여 새로운 컴포넌트(new component)를 만드는 것이 더욱 효과적인 방법이 된다.

3.2.8 기술적인 변경

컴포넌트 플랫폼 변경, 어댑터를 이용한 상호운용성(interoperability) 처리, 다중 데이터베이스 지원 또는 데이터 중복과 같은 기술적인 이슈를 해결함으로써 컴포넌트와 요구사항 사이에 격차를 제거한다.



(그림 2) 래퍼와 어댑터를 이용한 기존 코드 사용



(그림 3) 래핑을 위한 정제(refine) 로직

4. 컴포넌트 획득

컴포넌트를 어떻게 획득할 것인가를 결정한 후에는 컴포넌트 획득에 대한 승인을 얻기 위한 시안을 작성한다. 여기에는 다음의 내용을 포함하게 된다;

- 컴포넌트 식별 또는 획득에 대한 간략한 문제 기술서.
- 컴포넌트 획득에 대한 권고안.
- 컴포넌트의 선정 사유.
- 컴포넌트 혹은 업무 변경에 대한 내용. 선택한 컴포넌트이 단점. 솔루션을 적용하는 필요한 시간.
- 컴포넌트 구매, 변경, 설치 및 유지보수를 위해 필요한 예상 비용.
- 대안이 되는 다른 컴포넌트와의 비교.

5. 결 론

지금까지 체계적인 컴포넌트 획득절차를 통하여 CBD를 도입하기 위한 보다 실천적인 방안에 대해서 고찰해 보았다. 기존 시스템의 통합에서 처럼 컴포넌트 재사용은 하향식(top-down)뿐만 아니라 상향식(bottom-up) 접근법을 통해서도 이루어질 수 있으며, 이것이 보다 경제적인 방법이 될 것이다[3]. 연구 결과, CBD 접근법을 채택하기위해 필요한 조직적이고 체계적인 규칙은 달성하기에 어려울 수도 있음을 알았다. 만일 컴포넌트에 대한 적합성 분석이나 요구사항과 격차가 있는 컴포넌트에 대한 변경이 명세화되지 않는다면, CBD를 도입하려는 조직은 오히려 큰 혼란과 막대한 비용의 낭비를 초래하게 된다. 따라서 원하는 기능의 전부 혹은 일부를 갖는 컴포넌트를 식별하였을 경우 그것을 사용하기 위한 계획을 사전에 세우고 획득하기 위한 체계적인 프로세스가 필요하며, 본 논문에서 제시한 컴포넌트 획득절차가 이 경우에 사용될 수 있다.

6. 기 타

본 논문은 Katharine Whitehead와 협력하에 그녀의 저서 'Component-Based Development: Principle and Planning for Business Systems[4]'를 국내 실정에 맞게 소개한 것이다. 그녀는 현재 Critical Path사에서 CBD 분야의 컨설턴트로 활동 중이며, 지난 20년간 소프트웨어 아키텍처와 개발 방법론 분야의 세계적인 권유자로 알려져 있다.

참고문헌

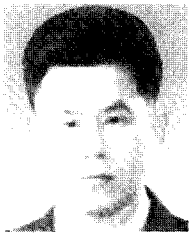
[1] Peter Herzum and Oliver Sims, "Business Component Factory: A Comprehensive Overview of Component-Based Development for the Enterprise", John Willey & Sons, Inc, 1999

[2] Jennifer Stapleton, "DSDM: Business Focused Development, Second Edition", Addison-Wesley, 2003

[3] 김경주, 조남규, "UML Components: 컴포

넌트 기반 소프트웨어 명세를 위한 실용적인 프로세스", 인터비젼, 2001
[4] Katharine Whitehead, "Component-Based Development: Principles and Planning for Business Systems", Addison-Wesley, 2002

저자약력



조 남 규

1998년 순천향대학교 컴퓨터공학과 (공학사)
2001년-현재 중앙대학교 컴퓨터소프트웨어학과 석사과정
1997년-2001년 Sterling Software Korea 컨설턴트
2001년-2002년 ComponentVision 선임컨설턴트
2003년-현재 SolutionLink 책임컨설턴트
관심분야: Component-Based Development, Product-Line Engineering, Software Architecture, Software Process Improvement