대규모 객체지향소프트웨어개발에 있어 설계산출물의 유용성

임 좌 상*

The Usefulness of Design Artifacts for Enterprise Object Oriented Software Development

Joa Sang Lim*

Abstract

This paper surveyed 30 system designers and developers who had been participating in a project for five months to examine the usefulness of design artifacts for object-oriented system development. A set of nine requirement and design documents was selected in consideration of the IEEE guidelines. Overall the respondents appeared to agree that the artifacts were useful for system development. On the other hand, the contribution of design artifacts to maintenance was rated lower due to the frequent changes over the iterative process. Of the artifacts, considered more useful were use case specifications, UI design and operation specifications. The respondents were the most reluctant to write test cases. Designers did not like to document more details in the design artifacts than did developers. A future study is required to determine the economics and change management of documentation.

Keywords : Software Design, Documentation, Object-Oriented Development, UML

논문접수일 : 2004년 10월 23일 논문게재확정일 : 2004년 11월 27일
※ 2004년도 상명대학교 교내연구비 지원에 의해 수행되었음.
* 상명대학교 소프트웨어대학 미디어학부 교수

1. 서 론

우리는 보통 소프트웨어를 개발하면서 실행파일을 생성하는 것에 많은 자원을 할당하고 문서를 작성하는 것은 소홀히 하는 경향이 있다. 여기서 문서는 설명서와 도움말과 같은 사용자 문서뿐만 아니라 소프트웨어의 개발과정에서 활용되는 설계산출물도 포함하는데, 본 연구는 설계산출물에 국한하여 '왜 문서화를 소홀히 하는 것일까?'에 대한 의문에서 출발한다. 문서의 중요성은 새삼 의심의 여지가 없는데, 소프트웨어 공학 교과서에서도 요구사항, 분석, 설계, 개발, 테스트, 전개 전 과정에 걸쳐 지식을 전달하는 수단으로 반드시 필요한 것이라고 강조하고 있다[Schach, 2004]. 그러나 프로그래밍이나 소프트웨어공학을 처음 배우는 학생뿐만 아니라 현장의 실무자가 시스템을 개발, 유지보수할 때도 문서는 통과의례로 간주해서 작성하는 것이 현실이다[Lim et al., 2004]. 이는 요구사항을 시스템화하기 위해 작성되는 설계문서가 개발에 도움이 되지 않거나, 도움이 된다고 하더라도 그 자원을 다른 활동(예 : 프로그래밍)에 투입하게 되면 더 유용할 것이라는 부정적인 태도에서 기인하는 것으로 추정된다. 그렇다면 문서화를 하지 않거나 줄이려고 하는 경향에 대한 해결책은 없는 것일까? 지금까지 문서산출물에 대해 이러한 행태적인 측면은 간과되어 온 경향이 있었다. 그 바탕에는 산출물은 '좋다-싫다'의 행태적인 문제라기 보다는 반드시 해야 하는 규범적인 성격을 띠고 있기 때문이다. 따라서 지금까지의 연구는 규범적인 관점에서 개발에 필요한 시스템적 요소와 이를 효과적으로 담을 수 있는 산출물의 형태를 찾아내는 측면에 초점이 맞추어져 있었다[Rumbaugh et al., 1991]. 본 논문에서는 행태적인 측면을 관찰하기 위해 현재 진행되고 있는 소프트웨어개발프로젝트에 참여하고 있는 설계자, 개발자를 대상으로 그들의 산출물에 대한 인식, 태도, 설계산출물의 유용성, 적정량, 상세수준에 관하여 조사하였다. 이러한 접근을 통해 본 연구는 소프트웨어개발의 중요한 일부분인 문서화에 대한 현황을 분석하고 그 문제점을 이해하는데 목적이 있다.

2. 소프트웨어설계와 문서화

소프트웨어는 프로그램, 소스코드나 실행파일뿐만 아니라 부속문서도 포함한다. 따라서 문서는 소프트웨어 품질의 중요한 속성으로 인정되고 있다[ISO9126, 1991]. 여기서 문서는 절차문서와 제품문서로 구분한다. 전자는 소프트웨어의 개발절차에 관련한 것으로 프로젝트계획 또는 프로그래밍 표준과 같은 지원성격의 문서가 해당된다. 후자인 제품문서는 소프트웨어 자체에 관한 것으로 사용자문서와 시스템문서로 구분한다. 사용자문서는 사용자매뉴얼과 같이 소프트웨어시스템(컴포넌트)의 사용법을 기술하는데, ANSI/ ANS 10.3-1995 표준에서 요약, 어플리케이션 정보, 문제기술과 프로그래밍 정보를 포함해야 한다고 제안하고 있다[Phoha, 1997]. 반면 시스템문서는 소프트웨어개발과정에 걸쳐 작성되는 문서로서 요구사항명세서, 분석명세서, 설계명세서, 프로그램소스, 테스트케이스 등이 포함된다. 문서화를 소홀히 한다는 것은 프로그램 소스코드를 제외한 설계문서를 적절하게 작성하지 않거나, 활용하지 않는다는 것을 의미한다. 물론 컴퓨터시스템이 등장한 처음부터 설계문서의 필요성이 인식된 것은 아니다. 1960년대 후반 객체지향이 처음 소개되었을 당시에 비해 시스템의 규모가 커지고, 분산화되면서 1990년대 초반, 보다 체계적인 개발방법론과 더불어 문서화의 필요성이 제기되었다[Rumbaugh et al., 1991]. 즉 1960~1970년대 대

형호스트의 중앙처리방식에서는 굳이 설계가 필요할 정도로 시스템이 복잡하지 않았다. 그러나 1980년대에 들어와 시스템이 분산되면서 객체를 분할하고 공유하는 문제, 분산된 객체 간의 통신문제와 더불어 시스템성능과 같은 복잡한 변수들이 소프트웨어개발에 큰 영향을 미치면서 설계를 보다 체계적으로 수행하고 이를 의사소통하기 위해 문서화가 필요하게 되었다. 이전까지의 프로그램 자체, 즉 소스코드와 실행파일에 초점을 맞추는 사고에서 벗어나 그 사전단계에 보다 많은 시스템적 사고가 필요하게 되면서 설계의 중요성이 부각되었고 이로 인해 문서화에 대한 필요성도 같이 강조되었다.

설계는 요구사항(업무지식)을 시스템으로 변환하는 문제해결과정으로 창의적인 사고가 필요하지만 '표현'과 '절차'를 통해 수행된다[Budgen, 2003]. 즉 시스템적 사고를 어떻게 표현할 것이며(예 : 데이터흐름도, UML) 이러한 표현간의 전환에 필요한 절차(예 : 유즈케이스기술서에서 클래스후보 인식)가 곧 설계인 것이다. 설계문서는 여러 관점에서 기술될 수 있는데, IEEE에서는 필수적인 4가지 요소로서 (1) 분할(decomposition), (2) 연관(dependency), (3) 인터페이스(interface), (4) 상세설계(detailed logic)를 권고하고 있다[IEEE, 1998b]. 첫 번째 '분할'은 전체시스템이 서브시스템, 클래스, 엔티티 등의 구성을 나타내는 것으로 계층적 분할도 또는 자연언어로 기술할 수 있다. 두 번째 속성인 '연관'은 데이터흐름도(DFD) 또는 클래스다이어그램과 같이 분할된 엔티티 간의 관계를 표현한다. 세 번째인 '인터페이스'는 분할된 엔티티를 사용하는 방식을 인터페이스목록, 파라미터목록으로 표현할 수 있다. 마지막으로 '상세설계'는 흐름도, 의사코드가 사용될 수 있다. 그간 수 많은 방법론과 표기법이 제안되면서 오히려 혼란이 초래되었지만, 최근에는 OMG의 UML(Unified Modeling Language)과 UP (Unified Process)로 설계표준이 통합되면서 정착되었다.

앞서 언급한 바와 같이 설계는 문서를 통해서 이루어지는 것이 보통인데, 실제 문서화에 대한 인식은 매우 미흡한 형편이며 이로 인해 후방의 개발에도 많은 영향을 미치고 있다[Leth bridge et al., 2003]. Visconti와 Cook[2004]은 시스템 개발문서에 대한 프로세스모델인 DPMM(Documentation Process Maturity Model)을 고안해서, 최근 91개 프로젝트를 대상으로 문서화 절차에 대해 설문조사를 한 결과 문서화가 매우 만족스럽지 못하다고 지적하고 있다. 프로젝트의 약 70%가 문서를 중요시하지 않았으며 나머지 약 30%에서만이 산출물이 반드시 필요한 것이라고 하였다. 그러나 산출물을 작성한다고 하더라도 그 품질과 일관성은 역시 의문이었다. 문서화를 경시하는 행태는 Taylor et al.[2002]에서도 보고되었는데 웹 개발프로젝트의 불과 1/3만이 설계문서를 작성하였다. 작성하는 설계문서도 계층도, 레이아웃, 흐름도, 스토리보드와 같이 IEEE가 제안하는 필수요소를 충분히 기술하지는 않았다. 또한 조사한 25개 프로젝트에서 불과 7개 만이 테스트를 공식적으로 실시하였다. Lethbridge, Singer와 Forward[2003]의 현장연구에서도 문서화가 적절하게 이루어지지 않고 있으며 변경에 대한 유지보수의 어려움으로 문서의 작성을 연기하거나, 복잡하고 어려운 문서는 작성하지 않고 회피하는 경향이 있다고 했다. Lindvall과 Runesson[1998]도 요구사항 변경에 따른 유지보수가 필요한 부분을 추적하는데 설계산출물은 역공학이 용이하지 않기 때문에 그 유용성이 적다고 주장하였다. 즉 프로그램의 사전단계로서 완전하고, 정확하고, 일관성 있는 요구사항을 기술하는 것과 고품질의 소프트웨어를 생산하기 위해 이러한 요구사항을 시스템으로 변환하는 설계산출물은 그 중요성에 비해 경시되는 경향을 보이고 있는 것이다. 본 연구에서는 대규

모시스템개발프로젝트를 대상으로 설계산출물의 유용성을 조사하였다. 앞서 제기한 '왜 문서화를 소홀히 할까?'라는 연구의문에 대해 이들의 역할을 개발자와 설계자로 구분해서 그 태도를 분석하였다. 문서를 경시하는 경향을 분석하기 위해 산출물의 복잡성, 변경용이성, 중복성과 정합성, 상세화의 필요성에 대하여 조사하였다. 그 결과 산출물의 필요성은 인정하는 반면 작성하는데 많은 시간이 걸리고 변경에 따른 유지보수가 어려운 것으로 나타났다. 이러한 태도는 프로젝트의 자원과 일정의 환경요소에 영향을 받고 있는 것으로 보인다. 본 연구는 대규모 프로젝트를 대상으로 상당기간 산출물을 작성하고 사용한 실제참여자를 대상으로 수행한 현장성이 의미가 있는데 그 연구방법은 다음 장에서 서술하기로 한다.

3. 연구방법

3.1 설 계

본 연구는 현장의 설계자, 개발자를 대상으로 설계산출물의 유용성에 대한 태도를 조사하는 것을 목적으로 했다. 산출물의 종류와 내용은 프로젝트의 성격과 환경에 따라 상이한 것이 보통이므로 자동차판매사의 시스템구현 프로젝트에서 일정 기간 동안 산출물을 작성해 온 참여자를 대상으로 설문을 통해 그들의 태도를 취합하였다. 프로젝트는 전사적인 규모로서 인사, 회계, 영업, 구매, 물류, 중고차, 거점, 할부, 채권 업무를 포함했고 그 규모는 16,637 기능점수에 달했다(<표 1> 참조).

〈표 1〉 업무별 기능점수 규모

공통	인사	회계	영업	구매	물류	중고차	거점	할부	채권
612	3,239	3,521	2,630	937	1,306	1,362	996	1,084	1,005

시스템은 반복적 방법론에 따라 총 51주 걸쳐 컴포넌트기반생산라인방식으로 구현되었다. 본 연구는 프로젝트의 23주가 지난 시점에(상세화 단계, 총 3차 반복 후) 설계자 15명과 개발자 15명을 대상으로 수행되었다. 즉 상세화는 아키텍처를 안정화하고 본격적인 소프트웨어 구현으로 전환하는 단계로서, 설문참여자는 현재 프로젝트에서 산출물 작성(개발) 경험이 충분해서 설문에 충실한 응답이 가능했다.

3.2 산출물

설계산출물은 <표 2>와 같이 요구사항, 분석, 설계, 테스트 과정에 걸쳐 총 9종을 선정하였고 IEEE의 기준에 적합하도록 (1) 분할, (2) 연관, (3) 인터페이스, (4) 상세로직을 포함하고 있다[7]. 클래스다이어그램과 순차다이어그램은 클래스와 클래스간 관계를 표현한다는 점에서 IEEE의 '분할'과 '연관'에 해당하는 문서였다. 컴포넌트목록과 오퍼레이션명세는 IEEE에서 권고한 바와 같이 설계산출물에 포함되어야 할 '인터페이스'와 '상세로직'을 기술하고 있다. 그 외 산출물인 유즈케이스기술서는 객체지향 프로젝트에서 보통 채택하는 산출물로서, 화면 및 테스트 관련 산출물은 화면이 있는 소프트웨어의 개발에 유용한 것으로 저자의 프로젝트 수행경험에 의거해서 선정하였다. 개별산출물이 아닌 아키텍처문서(아키텍처 스타일, 프레임워크, 컴포넌트모델), 지원관련 산출물(프로젝트관리, 형상변경관리, 개발/테스트 환경)은 제외하였다. 9종의 산출물과 약어는 <표 2>에 설명되어 있다(이후 본문에서는 약어를 사용함).

〈표 2〉 산출물목록과 주요내용

약 어	명 칭	내 용
UCD	유즈케이스기술서	시나리오, 업무로직, 화면명(화면로직 제외), 입출력정보(논리)
UID	화면설계	물리적인 사용자 화면 설계(레이블, 필드크기, 스타일 등)
UIW	화면로직설계	물리화면로직(이벤트 별 설명), 논리/물리 속성명, 메시지목록
SD	순차다이어그램	시나리오 별 유즈케이스 실현(클래스간 책임 분할)
CD	클래스다이어그램	클래스(컴포넌트) 관계, 가시성, 오퍼레이션
CL	컴포넌트목록	컴포넌트 목록(물리), 인터페이스 목록(물리)
OPS	오퍼레이션명세서	오퍼레이션의 사전/사후조건, 시그니쳐, 프로그램명세(SQL)
UTC	단위테스트케이스	화면 테스트케이스(화면로직 + 업무로직)
CTC	컴포넌트테스트케이스	컴포넌트 테스트 케이스(업무로직)

3.3 설 문

설문은 총 114개 항목으로 구성되었고, 각 산출물 별로 개발유용성, 유지보수유용성, 이해(작성)의 용이성, 변경 정도, 다른 산출물과 중복, 다른 산출물과 정합성, 다른 산출물 이해(작성)의 필요성, 상세화의 필요성에 대하여 물었다. 설문은 응답자의 역할, 업무, 경험을 묻는 항목이 포함되었다. 작성에 소요된 시간은 30분이 넘지 않도록 하였다.

3.4 분석방법

분석을 시작하기 전, 수집된 자료 중 오류 데이터가 있는지 확인하였다. 설문에 응한 참여자는 대부분 충실히 응답을 하여 오류 데이터는 없었다. 단, 응답하지 않은 항목은 분석에 포함하지 않았다. 분석은 현황을 알아보기 위한 서술 통계를 주로 이용하였다. 또한 설계자와 개발자의 주관적 인식의 차이는 t 분석으로 유의수준 0.05에서 윈도우용 SPSS 11.0을 사용해서 검증하였다.

<표 3>에서 보듯이 설문의 참여자는 객체지향 프로젝트에 평균적으로 약 19.9개월 참여한 경험이 있다고 응답하였다(개발자 17.8개월, 설계자 21.9개월). 시스템개발프로젝트 경험은 평균적으로 46.1개월이었는데, 설계자가(63.5개월) 개발자(26.1개월)보다 경험이 많았다.

〈표 3〉 응답자 특성

프로젝트경험	개발자	설계자	평 균
객체지향 개발	17.79	21.93	19.93
일반적 개발	26.08	63.47	46.11

4. 연구결과

4.1 유용성

산출물의 유용성은 (1) 개발유용성과 (2) 유지보수유용성 측면으로 구분하여 질문하였다. 우선 개발유용성에 관하여 분석하였다(<표 4> 참조). 전반적으로 화면과 프로그램명세 관련 산출물에 대한 개발유용성은 높게 평가되었다 - UID(4.20), UIW(4.13), OPS(4.11). 화면은 사용자와 협의하는 경우 매우 유용하다고 응답하였다. 유즈케이스에 대한 유용성도 높게 인식하였다(UCD : 3.73). 유즈케이스기술서는 개발하면서 많이 참조하지는 않지만 업무를 이해하는데 유용하다고 응답하였다. 이와 같은 UID, UIW, OPS, UCD 산출물에 대해서는 설계자와 개발자 모두

높게 평가하는 경향이 있었다. 그 중 UID에 대해서 설계자는 개발자보다 개발유용성을 유의적으로 높게 평가하였다($p < .01$). 그 이유는 화면개발이 화면디자이너에 맡겨져서 개발자가 그 중요성을 낮게 평가한 경향 때문이었다. 테스트케이스의 유용성은 낮게 평가하는 경향이 있었다 - CTC(2.33), UTC(2.89). 그 첫 번째 이유는 유용성에 대한 의문으로 테스트케이스는 제품의 기능성을 확인하는 용도로 필요한 것이어서, 프로그래밍 작업에 직접적으로 사용되지 않는다는 거부감이다. 둘째로는 테스트케이스를 프로그래밍 전에 모두 상세화하는 것이 어려워서 완전성이 결여되었다는 거부감이다. 마지막으로, 노력에 비해 그 효과가 적다는 경제성에 대한 의문이었다. 따라서 이와 같이 테스트케이스를 자세하게 작성하기 보다는 화면에서 임의로 테스트하는 것으로 충분하다고 선호하는 경향이 있었다. 또한 SD(2.85)의 유용성은 높게 평가하지 않았다. 이는 아키텍처(설계 프레임워크)에서 정해진 내용을 단순 반복으로 작성하면서 SD에서 원래 의도한 책임분할설계가 약했기 때문이었다. 또한 개발자 입장에서는 OPS만 있어도 충분히 개발이 가능하며 SD가 그다지 필요하지 않다는 의견이었다. 즉 SD, CD 등은 설계 과정의 부산물이라고 간주하는 경향이 있었다. OPS에 대하여 설계자는 작성하기 부담스러워 하는 반면 개발자는 상당히 필요한 기본적인 문서의 하나라고 평가하였다. 개발자는 OPS의 가장 큰 문제점으로 객체와 관계형테이블의 연결을 위한 데이터베이스 실행문의 정합성에 관해 예를 들어 설계자가 작성한 SQL문이 실제 수행하면 물리적인 오류가 많이 발생하는 문제를 꼽았다. 두 번째로 소프트웨어품질의 중요한 구성요소인 산출물의 유지보수유용성을 분석하였다(<표 4> 참조). 산출물의 유지보수유용성은 개발유용성보다 상대적으로 낮다고 평가되었다. 이는 개발자와 설계자가 시스템의 인도 후에 유지보수를 직접 하지 않을 가능성이 많기 때문에 낮게 평가한 것으로 보인다. 즉, UCD, UID, UIW, OPS, UTC 5종의 산출물 모두 개발유용성이 유지보수유용성보다 유의적으로 높다고 평가되었다. 예외는 CL로서 개발유용성 보다는 유지보수유용성이 높게 평가되었다. 그 이유로는 유지보수의 경우 구현의 상세한 내용을 은익해서 블랙박스 방식으로 사용할 컴포넌트, 패키지, 오퍼레이션과 시그니쳐 등 CL에서 제공되는 정보가 매우 유용하다고 판단하였기 때문이었다.

〈표 4〉 산출물의 유용성

	개 발 유 용 성				유 지 보 수 유 용 성			
	개발자	설계자	평 균	유의성	개발자	설계자	평 균	유의성
UCD	3.67	3.80	3.73	-	3.53	2.87	3.20	-
UID	3.87	4.53	4.20	p<.01	3.67	3.20	3.43	p<.05
UIW	4.00	4.27	4.13	-	3.67	3.40	3.53	p<.05
SD	2.67	3.00	2.85	-	2.75	3.07	2.93	-
CD	2.58	3.33	3.00	-	2.75	3.27	3.04	-
CL	2.73	3.80	3.35	-	2.91	4.13	3.62	p<.05
OPS	4.15	4.07	4.11	-	4.08	3.60	3.82	
UTC	2.69	3.07	2.89	-	2.46	2.36	2.41	-
CTC	2.46	2.18	2.33	-	2.62	1.91	2.29	p<.05

산출물 중에서는 OPS(3.82) CL(3.62), UIW (3.53), UID(3.43)와 같은 4종 산출물의 유지보수유용성이 높다고 평가되었다. 화면관련 문서인 UID, UIW는 개발자가 분석설계자보다 유의적으로 높다고 평가하였다($p < .05$). 그러나 유지보수 할 경우 실제화면으로 대체하는 것이 간편할 것이라는 의견이 있었다. 반면에 CL의 유지보수유용성은 설계자가 개발자 보다 높게 평가하였다($p < .05$). UCD는 시나리오를 설명하고 있어 업무담당자가 이직할 경우 유지보수에 유용할 것이라는 의견이었다. 테스트 관련 산출물은 앞서 개발유용성이 낮다고 했는데, 유지보수 유용성도 높지 않다고 평가하였다. 유지보수는 관련 산출물의 이해가 필수적인데, 설계산출물을 이해하는데 가장 큰 공헌을 한 문서는 UCD로서, 화면, SD, OPS의 이해에 도움이 많았다고 응답하였다. OPS를 이해하는데 UCD 뿐만 아니라 화면이 중요한 역할을 했다고 응답하였고, 테스트케이스문서는 다른 산출물 이해에 도움이 많이 되지 않는다는 의견이 많았다.

4.2 정합성과 변경빈도

산출물의 정합성과 변경빈도는 밀접한 관련이 있다. 특히 반복적 방법을 적용하게 되면 요구사항을 초기에 모두 추출할 수 없다고 가정하기 때문에 산출물의 변경관리가 필수적이고 정합성을 유지하는 것이 어렵게 된다. <표 5>는 설계자, 개발자 양측 모두 전반적으로 요구사항(UCD : 3.28), 화면(UID : 3.60, UIW : 3.63) 그리고 오퍼레이션명세서(OPS : 3.61)의 변경빈도가 높다고 응답한 결과를 나타내고 있다.

이는 사용자 요구사항의 변경으로 인해 발생한 회귀결함(regression faults)에서 비롯된 변경으로, 예를 들면, 이벤트 추가, 정보항목 변경, 목표업무프로세스의 미확정 등을 꼽을 수 있다. 따라서 이들 산출물의 정합성을 유지하는 것이 어렵다고 인식하는 경향이 있었다. <표 5>에서 보듯이 설계자, 개발자는 평균적으로 UCD(3.25)의 정합성을 유지하는 것이 어렵다고 응답했으며, 이는 화면설계의 변경과 밀접한 관련이 있는 것으로 보인다. 설계자는 UIW(3.31), OPS (3.27)의 정합성을 지키는 것이 어렵다고 응답하였는데, OPS는 요구사항문서인 UCD와의 정합성(26.7%), 시나리오를 동적으로 분석하는 SD와의 정합성(30%)을 유지하는 것이 어렵다고 응답하였다. 변경빈도에 대한 개발자와 설계자 간 유의적인 차이가 없었다. 특히 UCD($p < .05$), CTC($p < .05$)에 대해서는 설계자가 개발자 보다 정합성을 유의적으로 지키는 것이 어렵다고 응답하였는데, 이는 설계자가 산출물을 변경하는 당사자이므로 예민하게 반응한 것으로 보인다 (개발자는 소스코드의 변경책임.).

〈표 5〉 산출물의 변경빈도와 정합성

	변 경 빈 도			정 합 성		
	개발자	설계자	평 균	개발자	설계자	평균
UCD	3.21	3.33	3.28	3.23	3.27	3.25
UID	3.47	3.73	3.60	2.92	3.07	3.00
UIW	3.20	4.07	3.63	2.85	3.31	3.08
SD	3.00	2.80	2.88	2.42	2.80	2.63
CD	2.58	2.87	2.74	2.42	2.80	2.63
CL	2.45	2.57	2.52	2.60	2.29	2.42
OPS	3.54	3.67	3.61	2.75	3.27	3.04
UTC	2.50	3.00	2.76	2.33	2.93	2.65

4.3 중 복

<표 6>에서 보듯이, 개발자는 산출물 간의 중복에 대해 전반적으로 부정적인 의견을 보인 반면 설계자는 유즈케이스기술서, 테스트케이스가 중복된다는 의견을 보였다. 그러나 전반적으로 테스트케이스에 대해서는 다른 산출물과의 중복이 높다고 평가하지 않았다(UTC 2.38, CTC 2.83). 이는 앞서 설명한 바와 같이 낮게 평가된

테스트케이스의 유용성이 다른 산출물과 중복보다는 테스트 자체를 부가적인 작업으로 여기는 경향을 반영한 것으로 보인다.

〈표 6〉 산출물의 중복과 상세화

	중 복			상 세 화		
	개발자	설계자	평 균	개발자	설계자	평균
UCD	2.71	3.33	3.03	3.60	1.80	2.70
UID	2.36	2.27	2.31	3.36	2.20	2.76
UIW	2.53	2.73	2.63	3.31	2.07	2.64
SD	2.67	2.73	2.70	2.25	2.00	2.12
CD	2.58	2.67	2.63	2.18	2.14	2.16
CL	2.30	1.93	2.08	2.36	2.29	2.32
OPS	2.31	2.60	2.46	3.85	2.20	2.96
UTC	2.08	2.64	2.38	2.67	2.07	2.35
CTC	2.25	3.45	2.83	2.58	2.00	2.30

응답자는 산출물간 요구사항문서(UCD)와 설계문서(UID 60.0%, OPS 26.7%, UIW 23.3%)가 서로 중복된다고 응답하였다. 가장 많은 응답자가 지적한 UCD와 UID의 상호중복은 유즈케이스기술서의 논리적인 데이터항목이 화면설계에서 중복되어 표현되기 때문인 것으로 보인다. 다음으로 UCD와 OPS가 업무로직이 서로 중복되어서(UCD에는 논리적으로, OPS에서는 물리적으로) 기술된다고 응답하였다. 흥미롭게도 전반적으로 설계자가 중복을 더욱 느낀다고 응답하였는데, 그 이유는 산출물이 설계자에 의해 직접 작성되었고 개발자는 이를 활용하였기 때문에 설계자가 더욱 많은 중복을 지적하는 경향이 있었다. 이와 같이 요구사항과 설계산출물은 어느 정도 중복이 있는 것이 당연하지만, 설계문서 간의 중복은 거의 없다고 응답하였다. 설계문서 간에는 (1) CD와 OPS의 중복(40%), (2) SD와 OPS가 내용이 서로 중복되는 경향이 있다고 평가하였다(40%). 이는 OPS에서 책임설계방식에 필요한 오퍼레이션의 시그니쳐, 사전조건, 사후조건이 기술되었기 때문이었다.

4.4 상세화

<표 6>에서 보듯이 설계자, 개발자 양측 모두 산출물을 더욱 자세하게 작성하는 것에 대해서는 부정적이었다. 설계자가 개발자 보다 더욱 부정적인 경향이었으나 UCD($p < .05$)를 제외하고는 유의적이지 않았다. 즉, UCD에 대해서 개발자가 설계자보다 더욱 자세하게 적어야 한다고 응답하였는데, 특히 시스템반응, 화면내용, 입출력내용을 상세하게 적는 것이 바람직하다고 했지만(<표 7> 참조) 유의적이지 않았다.

〈표 7〉 유즈케이스기술서의 상세화

	개발자	설계자	평 균
크기의 적정성	2.87	2.86	2.86
시스템반응기술	3.80	2.33	3.07
화면내용기술	3.87	2.13	3.00
입출력항목기술	3.53	2.47	3.00

유즈케이스의 크기는 적정하다는 의견이 많았는데(2.86), 약 4개 내외의 기능을 갖고 있도록 설계되었다(기능점수 약 20점 정도). 그러나 UCD와 OPS의 업무로직이(UCD의 시나리오와 통제레이어를 기술하는 것이) 유사하기 때문에 상세하게 적는 것이 중복을 초래한다고 응답하였다. 설계자와 개발자 모두 OPS는 오퍼레이션별로 적는 것을 찬성하였다(3.61).

4.5 산출물 난이도

<표 8>에서 보듯이 개발자가 이해하기 가장 어려운 산출물은 OPS로서 평균 40분 정도, 17.68%의 시간이 소요된다고 응답하였다. 다음으로는 CTC(27.0분, 11.94%), UTC(27.1분, 11.98%)와 테스트관련 산출물이었는데, 합산하면 54.1분(23.92%)에 달하였다. 가장 쉬운 산출물은 SD(15.8분, 6.98%), CD(17.3분, 7.65%)로서 UML 문서였으며, 이는 그림으로 표현된 산출물이 이해하기

용이하다는 것을 반증하고 있다. 설계자 역시 개발자와 유사한 응답을 보여주었다. 설계자가 작성하기 가장 어려운 산출물은 OPS(383.3분, 22.93%), CTC(282.9분, 16.92%)였다. 양측 모두 SD(71.3분, 4.27%)와 CD(42.7분, 2.55%) 산출물은 상대적으로 적은 시간이 소요되었다고 했다. CL은 작성하는 시간이 비교적 적다고 응답하였는데(47분, 2.81%) 이 산출물은 아키텍트에 의해 일괄적으로 작업된 것을 수정하기 때문에 쉽게 작성되었다. 이러한 작업시간은 프로젝트에서 분석된 일일 작업시간과 많은 차이가 없었다(<표 8> 세 번째 열).

이러한 작성(이해)시간을 설계자 별로 분석하면 개인숙련도에 따라 매우 큰 편차를 나타냈다(최대 8배). 프로젝트의 설계, 구현에 소요된 공수는 기능점수 당 약 19.6시간으로 분석되었으며(설계 7.4, 구현 12.2시간) 지원에 투입된 공수(관리, 형상관리, 아키텍트 등)를 감안하면 이보다 높다고 추정된다(<표 9> 참조).

〈표 8〉 산출물의 작성(이해) 시간

	개 발 자	설 계 자	작업시간
UCD	28.5 (12.60%)	222.9 (13.33%)	158.4 (9.33%)
UID	26.2 (11.58%)	205.3 (12.28%)	481.2(28.33%)
UIW	24.3 (10.74%)	202.1 (12.09%)	
SD	15.8 (6.98%)	71.3 (4.27%)	192.0(11.30%)
CD	17.3 (7.65%)	42.7 (2.55%)	
CL	20.0 (8.84%)	47.0 (2.81%)	NA
OPS	40.0 (17.68%)	383.3 (22.93%)	474.0(27.91%)
UTC	27.0 (11.94%)	214.2 (12.81%)	393.0(23.14%)
CTC	27.1 (11.98%)	282.9 (16.92%)	
합계	226.2(100.00%)	4,271.7(100.00%)	100.00%

〈표 9〉 단계별 투입공수와 생산성

	투입공수	생 산 성
설 계	5,400 시간	7.40
구 현	3,240 시간	12.23

4.6 산출물 결함

모든 설계산출물은 작성된 후 검토자에 의해

〈표 10〉 설계산출물의 검토항목과 결함빈도

결 함 내 용	빈도
Java 명명이 올바르게 되었습니까? (속성 및 오퍼레이션 명명규칙 포함)	22
UCD에 기술된 입출력정보의 명칭이 화면에 표현된 명칭과 (내용상) 일치합니까?	18
업무 요건상 화면의 사용자이벤트에 누락된 것이 없습니까?	18
UCD에 기술된 모든 입출력정보항목이 (내용상) 화면에 누락되지 않았습니까(필수항목이 모두 표현)?	16
데이터모델을 이용하여 데이터객체(DOMap, TDO)가 올바르게 생성되었습니까?	16
사용자이벤트의 화면로직 설명이 To-Be모델과 비교해서 부정확한 내용이 없습니까? (UCD의 서버로직 제외)	15
OPS의 처리흐름이 구현하기에 충분하도록 작성되었습니까?	14
UCD의 시나리오 수행에 필요한 모든 화면이 작성되었습니까?	14
업무 요건상 화면요소가 업무처리에 효율적으로 배치되어 있습니까?	13
포함유스케이스가 올바르게 SD와 CD에 사용되었습니까?	12
다른 컴포넌트에 존재하는 기능이 중복개발 되지는 않습니까?	10
리스트박스 또는 콤보박스와 같이 화면에 감추어졌지만 필요한 데이터를 채우는 방식이 충분하게 기술되었습니까?	10
UCD의 시나리오와 화면흐름이 동일합니까?	10
작성한 항목 가운데 문법, 표현법에 맞게 작성되었습니까?(예:필드유형)	10
체크버튼, 라디오버튼과 같은 화면요소에 필요한 데이터가 정확하게 기술되었습니까?	9
기술한 내용이 화면로직에서 사용한 모듈을 개발하기에 충분합니까?	8
리스트박스 또는 콤보박스와 같이 필요한 데이터가 정확하게 기술되었습니까?	6
UCD의 액터 이벤트가 SD에서 액터에서 출발하는 메시지와 일치합니까?	6
OPS의 설명/사전/사후 조건이 누락되지 않았습니까?	6
UCD와 UD에서 인식한 화면과 비교해서 차이가 없습니까?	5

서 품질을 평가 받았으며 불완전한 문서는 재설계가 요청되었다. 결함항목을 빈도 수로 분석하면 <표 10>과 같다. 가장 빈번하게 지적된 내용은 일관성(Java 명명오류 22건, 입출력정보명칭오류 18건)이었다. 또한 요구사항 누락(화면이벤트 누락 18건, 유즈케이스기술서 입출력정보 누락 18건), 산출물의 부 정확성(클래스 부정확 16건, 화면설명 부정확 15), 산출물의 완전성(업무로직의 불완전 14건, 화면의 완전성 14건)도 빈번하게 지적되었다. 설계 산출물을 작성하면서 발견된 결함밀도는 기능점수 10점당 0.96으로 측정되었는데 이는 약 2시간의 수정작업이 필요한 정도의 결함수준이었다.

5. 결 론

소프트웨어의 설계표준이 UML로 정착되고, 객체지향이 확산되면서 컴포넌트를 기반으로 한 다양한 개발방식이 등장하고 있다. 특히 생산라인방식의 패러다임은 소프트웨어의 생산자와 소비자를 직능화하면서 설계산출물을 직능간 효과적인 의사소통수단으로 강조하고 있다[Atkinson et al., 2002 ; Clements & Sametinger, 1997]. 그러나 산출물에 대한 개발자의 태도와 이를 어떻게 활용하는지에 대한 연구는 많지 않다[Lethbridge et al., 2003]. 본 연구는 실제 프로젝트에 참여하고 있는 설계자와 개발자를 대상으로 생산자에게 제공되어야 할 문서에 대하여 실태조사를 하였다. 우선 생산자에게 필수적인 산출물 IEEE 1016-1998 표준에 따라[IEEE, 1998a] 9종을 선정하고 이들 산출물에 관하여 유용성 및 사용성에 관하여 물었다. 그 결과 제시된 9종의 산출물 가운데 유즈케이스기술서, 화면, 프로그램명세가 시스템개발에 유용한 것으로 나타났다. 그러나 산출물 작성에 많은 시간이 필요하며, 변경이 자주 발생해서 유지보수하기 어렵다는 인식을 보였다. 이러한 결과는 Lethbridge, Singer과 Forward[2003]과 일치하고 있다. 변경은 반복적 방식이 적용되면서 필수적으로 수반되는데 특히 요구사항이 불안정(목표 업무프로세스 미확정)하거나 설계가 변경되는 것(데이터항목 명)이 주된 원인이었다. 잦은 변경으로 인해 이들 산출물의 유지보수 유용성은 상대적으로 낮게 평가되었다. 사용자의 관점에서 컴포넌트를 사용할 때 필요한 컴포넌트와 인터페이스목록이 유지보수에 유용할 것으로 인식되었다. 특히 유즈케이스기술서는 설계산출물과 중복되는 경향이 있지만 설계산출물을 이해하는데 많은 공헌을 하는 것으로 응답하였다. 동적 모델링에 필요한 순차다이어그램은 그다지 유용하지 못하다고 평가하였다. 이는 생산라인기법을 적용하면 상세설계가 아키텍처를 근간으로 상당부분 결정되는 경우가 많아서 순차다이어그램의 창의적인 역할이 제한적이기 때문인 것으로 추정된다. 테스트케이스에 대해서는 모두 부정적으로 평가하였는데, 그 역할이 컴포넌트의 품질을 확인한다는 관점에서 생산라인기법에서는 오히려 강조되어야 할 산출물이다. 설계산출물 작성에 걸리는 시간은 기능점수당 약 7시간이 소요되었으며, 전반적으로, 오퍼레이션명세서, 테스트케이스, 화면설계에 약 1/4씩의 시간이 투입되고 있었다. 설계산출물을 더욱 자세하게 작성하는 것에 대해서는 모두 부정적이었고, 오히려 설계산출물을 줄일 경우 개발이 빨라질 수 있을 것으로 응답하는 경향이 있었다. 이는 설계와 개발의 이중성을 보여주는 것으로 설계의 역할에 대한 재조명이 필요하다. 특히 흥미로운 것은 요구사항문서를 작성하면서 후방 설계문서를 참조한다는 점이었다. 즉 유즈케이스기술서를 작성하면서 입출력정보를 적기 위해 테이블설계서를 참조하는 경향이 있었다. 유즈케이스기술서는 그 목적이 사용자의 요구사

항을 시나리오를 중심으로 기술한다는 점을 감안해, 화면이나 테이블설계서의 모든 속성 보다는 주요한 정보항목을 위주로 작성해서 변경에 취약하지 않도록 하는 것이 바람직하다. 또한 유즈케이스기술서는 논리적으로 작성하고 논리와 물리의 완충역할을 하는 산출물을 작성하는 것이 바람직하다(예 : 용어집에서 한글명과 해당 영문 명을 기술). 유즈케이스기술서에는 초기에 요구사항을 정확하고 자세히게 파악하기 어렵다는 이유로 업무로직이 부실하게 적는 경향이 있었는데, 이를 위해 기존의 시스템이나 업무규정을 적절하게 활용하여 상세화하는 것이 필요하다. 테스트케이스는 작성공수에 비해 유용성을 낮게 평가하였지만 필수적인 산출물임에는 의심이 없다. 따라서 테스트케이스를 자동화하고 통계적인 기법을 활용해서 개발자의 노력을 줄여주는 공학적 접근이 필요하다.

요약하면 본 연구에서 설계자와 개발자 모두 산출물의 필요성은 정도간에 차이는 있지만 모두 인정하고 있었다. 그러나 작성노력에 비해 그 효과와 잦은 변경에 따른 유지보수성을 문제점으로 지적하고 있다[Lethbridge et al., 2003 ; Lindvall & Runesson, 1998]. 설계문서가 개발에 보다 적극적으로 활용되기 위해서 소프트웨어개발자의 인지노력을 최소화하는 방안이 필요하다. 설계는 복잡한 문제해결과정이고 따라서 설계자는 단순전략이나 회피전략을 택하여 문제를 가급적 단순화하는 것으로 심리학 연구에서 보고되고 있다. 예를 들어 변경에 있어, 유지보수를 용이하게 할 수 있도록 설계와 개발을 일치시켜는 역공학이 가능한 CASE 도구를 활용하면 설계자가 회피전략을 택하지 않고 변경을 충실하게 수행할 수 있을 것이다. 또한 산출물을 작성하면서 프로그래밍을 할 수 있는 연구가 시도되고 있는데 이러한 방안도 변경을 용이하게 해주는 도구로 주목할 필요가 있다(예 : literate programming, object-oriented documentation, [Childs & Northrop, 2002]). 또한 단순한 의사소통 도구에서 벗어나 설계를 충실하게 수행하게 되면 (비용) 후방개발에서 프로그램의 재사용성을 높일 수 있는 것인지, 또한 개발에서의 결함을 낮출 수 있는 것인지 (효과)에 대한 비용-효과적인 연구가 필요하다. 본 논문에서와 같이 현장연구에서는 많은 변수가 개입하므로 실험실에서 보다 통제된 연구방법을 통해 검증하는 후속연구가 필요하다. 이러한 연구를 통해 보다 효과적이고 고품질의 소프트웨어를 생산하기 위한 대안이 강구될 수 있을 것이다.

참 고 문 헌

[1] Atkinson, C., Bayer, j., Bunse, C., Kamsties, E., Laitenberger, O., Laqua, R., Muthig, D. Paech, B., Wust, J., and Zettel, J., Component-Based Product Line Engineering with UML, Addison-Wesley, Sydney, 2002.

[2] Budgen, D., Software Design, Addison-Wesley, Sydney, 2003.

[3] Childs, B., and Sametinger, J., Analysis of literate programs from the viewpoint of reuse, Software-Concepts and Tools, Vol. 18, No. 1, 1997, pp. 35-46.

[4] Clements, P., and Northrop, L., Software Product Lines : Practices and Patterns, Addison-Wesley : Sydney, 2002.

[5] Gellevij M., and Van Der Meij H., Empirical proof for presenting screen captures in software documentation, Technical Communication, Vol. 51, No. 2, pp. 224-238.

[6] IEEE, IEEE guide to software design descriptions, IEEE Std 1016.1-1993, The Institution of Electrical and Electronics Engineers, Inc., 1993.

[7] IEEE, IEEE recommended practice for soft-

ware design descriptions, IEEE Std 1016-1998, The Institution of Electrical and Electronics Engineers, Inc., 1998a.

[8] IEEE, IEEE standard for software test documentation, IEEE Std 829-1998, The Institution of Electrical and Electronics Engineers, Inc., 1998b.

[9] IEEE, IEEE standard for software user documentation, IEEE Std 1063-1987, The Institution of Electrical and Electronics Engineers, Inc., 1988c.

[10] ISO 9126, Software Product Quality Characteristics, http://www.cse.dcu.ie/essiscope/sm2/9126ref.html

[11] Lethbridge, T.C., Singer, J., and Forward, A., How Software Engineers Use Documentation : The State of the Practice, IEEE Software, 2003, pp. 35-39.

[12] Lindvall, M., and Runesson, M., "The Visibility of Maintenance in Object Models : An Empirical Study", The Proceedings of the International Conference on Software Maintenance, 1998, pp. 54-62.

[13] Lim, J.S., Jeong, S.R., Whang, M.C., and Cho, Y.C., Practical Investigation into the Maintainability of Object-Oriented Systems for Mission Critical Business, Lecture Notes in Computer Science, Vol. 3261, 2004, pp. 554-563.

[14] Phoha, V., A standard for software documentation, IEEE Computer, Vol. 30, No. 10, 1997, pp. 97-98.

[15] Rumbaugh, J., Blaha, J., Premerlani, W. Eddy, F., and Lorensen, W., Object-Oriented Modeling and Design, Prentice-Hall, 1991.

[16] Schach, S.R., Object-Oriented Analysis and Design with UML and the Unified Process, Addison-Wesley, 2004.

[17] Taulavuori A., Niemela, E., and Kallio, P., Component documentation - a key issue in software product lines. Information and Software Technology, Vol. 46, No. 8, 2004, pp. 535-546.

[18] Taylor M.J., McWilliam, J., and Forsyth, H., et al., Methodologies and website development : a survey of practice, Information and Software Technology, Vol. 44, No. 6, 2002, pp. 381-391.

[19] Visconti, M., and Cook, C.R., Assessing the state of software documentation practices, Lecture Notes in Computer Science, Vol. 3009, 2004, pp. 485-496.

저자소개

임 좌 상

현재 상명대학교 소프트웨어대학 미디어학부에 부교수로 재직하고 있다. 지금까지 DSS, Journal of Behavioral Decision Making, International Journal of Forecasting, Human Factors, Journal of Systems and Software, Lecture Notes in Computer Science에 학술논문을 발표하였고, 주로 금융분야 정보시스템 구축 프로젝트에 참여하고 있다. 관심분야는 컴포넌트소프트웨어, 감성컴퓨터 등이다.

◈ 이 논문은 2004년 10월 23일 접수하여 1차 수정을 거쳐 2004년 11월 27일 게재확정되었습니다.