# 유스케이스 트랜잭션 기반의 소프트웨어 공수 예측 기법

# (Software Effort Estimation based on Use Case Transaction)

이 선 경 † 강 동 원 ††
(SunKyung Lee) (DongWon Kang)

배 두 환 †††
(Doo-Hwan Bae)

**요 약** 본 논문에서는 기존 유스케이스 점수 기법의 공수 예측 정확도 향상을 위해 유스케이스 트랜잭션을 기반으로 한 공수 예측 기법을 제안한다. 유스케이스 점수 기법은 소프트웨어 유스케이스 모델을 기반으로 하는 공수 예측 기법으로서 객체 지향 소프트웨어 개발 프로젝트에서 사용되고 있다. 그러나 유스케이스 점수는 트랜잭션의 개수를 규모 산정의 단위로 활용하여 트랜잭션 별 구현 공수의 차이를 반영할 수 없고 트랜잭션 수의 범위에 따라 유스케이스의 규모를 결정함으로써 상이한 트랜잭션 수를 갖는 유스케이스들이 공수 예측 시 동일한 크기로 반영되어 상세수준에서의 문제를 갖는다. 이런 한계점들은 부정확한 공수 예측을 야기하여 프로젝트의 성공률을 저해하는 요소가 될 수 있다. 이를 개선하기 위해 본 논문에서는 공수 예측 시 트랜잭션을 단위 연산으로 세분화하고, 각 연산에 대한 복잡도를 활용하여 규모를 산정하는 트랜잭션 점수 기법을 제안하고자 한다.

**키워드** : 유스케이스 점수, 공수 예측, 유스케이스 트랜잭션

***Abstract*** *Use Case Point(UCP) is a measure of a software project size for software effort estimation based on use case. UCP measures the size of the software project based on the use case model. Because UCP is based on the use case model, it is intuitive and easy to obtain. Also, it does not require extra artifacts. On the other hand, UCP has some problems. UCP assumes every transaction has the same complexity. But, the number of operations and complexity of operations may affect complexity of transaction. In addition, UCP uses simple rating scale of complexity, but it may be inadequate for detailed estimates. To solve these problems, we suggest "Transaction Point(TP)", a size measure based on use case transaction. TP considers actors and operations in transaction. Complexity of transaction is based on the number of operations and complexity of operation, so it can support detailed estimation.*

**Key words** : Use Case Point, software effort estimation, use case transaction

## 1. 서 론

유스케이스 점수(Use Case Point) 기법은 소프트웨어 유스케이스 모델을 기반으로 하는 공수 예측 기법으로서[3,4] IBM Rational software, Galorath, ISBSG, sd&n 등 객체지향 소프트웨어 개발 프로젝트에서 널리 활용되고 있다. 유스케이스 점수 기법에 관한 일련의 연구들은 유스케이스 점수 기법이 전문가에 의한 예측보다 비교적 정확하다는 결과를 보여주고 있다[1,2]. 유스케이스 점수 기법은 소프트웨어 개발 과정에서 사용되는 유스케이스 모델을 기반으로 함으로써 공수 예측을 위해 소요되는 추가적인 노력을 줄일 수 있는 장점을 지니며 또한 비교적 직관적이고 계산 과정이 쉽다.

그러나 위와 같은 장점에도 불구하고 유스케이스 점수 기법은 다음과 같은 문제점을 가지고 있다. 먼저 각 트랜잭션은 수행 연산의 종류 및 개수에 따라 구현에 소요되는 노력이 상이하나, 현재의 유스케이스 점수 기법에서는 트랜잭션의 개수를 규모 산정의 단위로 활용하여 트랜잭션 별 구현 공수의 차이를 반영할 수 없다는 한계를 지닌다. 또한 유스케이스 점수 기법은 트랜잭션 수의 범위에 따라 유스케이스의 규모를 결정함으로써 상이한 트랜잭션 수를 갖는 유스케이스들이 공수 예측 시 동일한 크기로 반영되어 상세수준에서의 문제를 갖는다. 위와 같은 문제점들은 부정확한 공수 예측을 야기하여 소프트웨어 개발 프로젝트의 성공률을 저해하는 요소가 될 수 있다.

이를 개선하기 위해 본 논문에서는 공수 예측 시 트랜잭션을 단위 연산으로 세분화하고, 각 연산에 대한 복잡도를 활용하여 규모를 산정하는 트랜잭션 점수(Trans-

· 본 연구는 방위사업청과 국방과학연구소의 지원으로 수행되었습니다. (UD060048AD)
· 이 논문은 제36회 추계학술발표회에서 '유스케이스 트랜잭션 기반의 소프트웨어 공수 예측 기법'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : KAIST 전산학과
sklee@se.kaist.ac.kr
†† 비 회 원 : KAIST 전산학과
dwkang@se.kaist.ac.kr
††† 종신회원 : KAIST 전산학과 교수
bae@se.kaist.ac.kr

논문접수 : 2009년 12월 18일
심사완료 : 2010년 2월 11일

action Point, TP) 기법을 제안하고자 한다. 트랜잭션 점수 기법은 공수 예측 시에 주어지는 동일한 정보를 바탕으로 보다 세밀한 공수 예측이 가능하므로, 기존의 유스케이스 점수 기법에 비해 높은 정확성을 보일 수 있는 장점이 있다.

본 논문의 구성은 다음과 같다. 먼저 2장에서는 본 연구의 배경이 되는 유스케이스 점수 기법에 대해 알아보고 3장에서는 유스케이스 점수 기법 및 관련 연구들을 설명하고 문제점을 분석한다. 4장에서는 본 연구에서 제안하는 트랜잭션 점수를 설명하고 5장에서는 실험을 통해 기존 유스케이스 점수 기법과 본 연구에서 제안된 트랜잭션 점수 간의 정확도를 비교 분석한다. 마지막으로 6장에서는 본 연구의 결론 및 향후 연구 방안에 대해 설명하겠다.

## 2. 배경지식

유스케이스 점수(UCP) 기법은 1993년 Karner에 의해 기능점수(Function Point, MK II Function Point) [6]로부터 확장된 공수 예측 기법으로 유스케이스 모델을 기반으로 개발 공수를 예측한다.

유스케이스 점수 기법은 유스케이스 모델에 기술된 액터와 유스케이스를 통해 조정 전 유스케이스 점수(Unadjusted Use Case Point)를 계산하고 시스템의 규모에 영향을 미칠 수 있는 시스템의 기술적 특성 및 프로젝트의 특성을 반영하기 위해 조정 전 유스케이스 점수에 기술적 복잡도 인자(Technical Complexity Factor) 및 환경 인자(Environmental Factor)를 활용하여 유스케이스 점수(Use Case Point)를 계산한다. 그리고 조직의 생산성(Person-Hour per UCP)을 반영하여 개발 공수를 예측한다. 구체적인 유스케이스 점수 계산 방법은 아래 표 1과 같다.

단계 4에서 기술적 복잡도 인자(TCF)는 총 13개 항목으로 이루어져 있다. TCF는 기술적 복잡도와 관련된 인자로 Albrecht의 연구[8]로부터 확장되었다. 시스템의 분산환경 정도, 코드의 재사용성 정도, 변경 용이성 정도 등이 그 예이다. 환경 인자(EF)는 프로젝트의 프로세스 및 자원의 특성과 관련된 인자이다. 객체지향 시스템 개발 경험 유무, 정규 근무 여부, 충분한 개발 동기 보유 여부 등이 이에 해당한다.

단계 6에서 개발 공수를 예측하기 위해 사용되는 생산성은 단위 유스케이스 점수(UCP) 규모의 시스템을 개발하기 위해 요구되는 공수(Person-Hour per UCP)이다. 연구 결과에 의하면 일반적으로 생산성 인자는 20~36 PHperUCP의 값을 가진다고 알려져 있다[3,9]. 그러나 보다 정확한 공수 예측을 위해서는 데이터 수집을 통해 각 조직의 프로젝트 환경을 반영한 생산성을 활용하는 것이 바람직하다.

표 1 유스케이스 점수 계산

| 단계 | 계산 방법 | 계산식 |
|---|---|---|
| 1 | · 액터의 분류에 따른 규모 계산<br>a) 다른 시스템(API 정의)<br>Simple, WF(Weight Factor) = 1<br>b) 다른 시스템(프로토콜정의),<br>사람(콘솔 사용)<br>Average, WF = 2<br>c) 사람(GUI 사용)<br>: Complex, WF = 3 | Unadjusted Actor Weights(UAW) = Σ(# of Actors * WF) |
| 2 | · 유스케이스 분류에 따른 규모 계산<br>a) 3개 이하의 트랜잭션<br>: Simple, WF = 5<br>b) 4~7개의 트랜잭션<br>: Average, WF = 10<br>c) 8개 이상의 트랜잭션<br>: Complex, WF = 15 | Unadjusted Use Case Weights (UUCW) = Σ(# of Use Cases * WF) |
| 3 | · 조정 전 유스케이스 점수(UUCP) 계산 | UUCP = UAW + UUCW |
| 4 | · 조정 인자 계산<br>1) 13개의 기술적 복잡도 인자(TCF) 및 8개의 환경 인자(EF)에 0~5 사이의 값을 할당한다.<br>2) 할당된 값에 가중치를 곱한다(가중치 : -1~2)<br>3) 기술적 복잡도 인자와 환경 인자를 계산한다. | Technical Complexity Factor(TCF) = 0.6 + (0.01 * TFactor)<br>Environmental Factor(EF) = 1.4 + (-0.03 * EFactor) |
| 5 | · 조정 인자를 반영하여 유스케이스 점수(UCP) 계산 | UCP = UUCP * TCF * EF |
| 6 | · 생산성을 반영하여 공수 계산 | Effort = UCP * PHperUCP |

## 3. 관련연구

최초로 유스케이스 점수 기법을 제안한 Karner[3]는 유스케이스의 규모를 측정하기 위해 유스케이스 모델 내의 액터와 유스케이스를 사용하였다. 이 중 유스케이스의 규모는 트랜잭션의 수에 따라 복잡도를 3단계(낮음, 보통, 높음)로 구분하고 복잡도에 따라 각각 5 10, 15의 가중치를 부여하였다. 그러나 이와 같이 트랜잭션 수의 범위에 따라 유스케이스의 규모를 결정하는 낮은 수준의 기준 상세화는 상이한 트랜잭션 수를 갖는 유스케이스들이 공수 예측 시 동일한 크기로 반영되는 문제를 갖는다. 예를 들어 두 개의 유스케이스가 각각 트랜잭션 4개, 트랜잭션 7개를 포함하고 있는 경우, 두 유스케이스는 트랜잭션 수가 다름에도 불구하고 10이라는 같은 복잡도를 가진다고 판단된다. 또한 유스케이스 점수 기법에서 각 트랜잭션은 수행 연산의 종류 및 개수에 따라 구현에 소요되는 노력이 상이하나, 현재의 유스케이스 점수 기법에서는 트랜잭션의 개수를 규모 산정의 단위로 활용하여 트랜잭션 별 구현 공수의 차이를 반영할 수 없다는 한계를 지닌다. 예를 들어 단순히 사

용자 요청에 대한 시스템 응답을 주는 트랜잭션에 비해 사용자 요청에 대해 계산 및 검증, 데이터 저장 및 조회 후 시스템 응답을 주는 트랜잭션은 보다 많은 개발 공수를 요구할 수 있다. 또한 같은 연산 단위라고 해도 그 복잡도에 따라 서로 다른 개발 공수를 요구할 수 있다. 예를 들어 간단한 계산기 응용 프로그램 내의 트랜잭션에서 수행되는 사칙연산 연산에 비해 전투기 시뮬레이션에서 타겟 추적을 수행하는 트랜잭션 내 계산 연산이 훨씬 더 큰 복잡도를 가진다. 따라서 복잡도의 단위로서 트랜잭션을 사용하는 것은 부정확한 공수 예측의 하나의 원인이 될 수 있다.

Marcio[10]은 유스케이스 점수 기법이 유스케이스 내의 복잡도를 정확하게 반영하지 못하는 문제점을 해결하기 위해 액터와 유스케이스 외에 유스케이스 내의 엔티티의 수, 유스케이스 조건(Pre-condition, Post-condition) 내의 논리적 표현(logical expression)의 수를 추가적으로 고려한 유스케이스 규모 점수(Use Case Size Point, USP)를 제안하였다. 유스케이스 규모 점수는 액터의 종류, 엔티티의 수, 유스케이스 조건 내의 논리적 표현 수에 따라 복잡도를 분류하고 이에 따른 가중치를 부여하였다. 그러나 유스케이스 규모 점수는 유스케이스 모델에서 엔티티를 추출하는 과정이 추가적으로 요구된다. 또한 유스케이스 규모 점수는 엔티티의 수, 논리적 표현의 수, 트랜잭션 수의 범위에 따른 유스케이스의 규모 결정 방식을 사용함으로써 기존 유스케이스 점수 기법의 문제점 중 하나인 낮은 수준의 기준 상세화 문제점을 해결하지 못했고 트랜잭션 간의 상이한 복잡도 역시 반영하지 못했다.

반면 본 논문에서 제시한 트랜잭션 점수는 트랜잭션을 단위 연산으로 세분화하고, 각 연산에 대한 복잡도를 활용하여 규모를 산정함으로써 기존 Karner의 연구가 가지는 상세화 문제를 해결하였고, 또한 단위 연산의 복잡도를 통해 전체 시스템의 규모를 산정하기 때문에 트랜잭션마다 가지는 복잡도의 차이를 반영할 수 있다. 또한 유스케이스 모델 외에 추가적인 데이터를 요구하지 않는다는 점에서 Marcio의 연구와도 차별화된다.

## 4. 트랜잭션 점수 기법

본 논문에서는 트랜잭션의 복잡도에 영향을 미치는 트랜잭션의 단위 연산을 정의하고, 이 단위를 통해 유스케이스의 복잡도를 측정하고자 한다. 이 장에서는 트랜잭션을 구성하는 단위 연산을 정의하고 단위 연산의 수 및 복잡도를 활용하여 공수를 예측하는 트랜잭션 점수(Transaction Point, TP)를 설명한다.

### 4.1 트랜잭션의 단위 연산 정의

트랜잭션을 구성하는 단위 연산의 분류 및 복잡도를 정의하기 위해 본 논문에서는 COCOMO II[5] 모델의 시스템 구성 연산요소를 활용하였다.

COCOMO II 모델은 시스템 및 컴포넌트의 특성을 결정할 수 있는 다섯 가지 연산의 종류를 아래와 같이 정의하고 각 연산의 복잡도에 따른 가중치를 활용하여 개발 비용을 예측한다.

- 통제 연산(control operations)
- 계산 연산(*computational operations*)
- 장치 의존적 연산(device-dependent operations)
- 데이터 관리 연산(data management operations)
- 사용자 인터페이스 연산(UI management operations)

그러나 COCOMO II 모델의 다섯 가지 연산은 설계가 완료된 시점에 식별 가능한 것으로 유스케이스 모델에서는 식별이 어렵다. 이 문제를 해결하기 위해 Lieberman[7]의 설계 메커니즘(architectural relevance)의 특성(characteristics)을 활용하였다. Lieberman은 시스템 개발 시 위험관리를 위해 설계 메커니즘을 유스케이스에서 식별하고 이를 통해 구현 난이도를 측정하여 비교적 구현 난이도가 어려운 유스케이스를 먼저 구현함으로써 일정 연기 및 비용 증가의 위험을 줄이고자 하였다. 이때 기술 수준이 다른 설계 메커니즘을 유스케이스에서 식별하기 위해 설계 메커니즘에 주요 특성을 정의함으로써 유스케이스 내에서 설계 메커니즘을 식별할 수 있는 가이드를 제시하였다.

트랜잭션 단위 연산의 분류, 복잡도, 가중치 및 구체적인 분류 가이드는 아래 표 2와 같다. 아래 복잡도 분류 가이드에 정확히 대응되지 않는 경우에는 가장 유사한 분류의 복잡도를 활용한다.

COCOMO II 모델의 복잡도에 따른 가중치를 적용하여 단위 연산의 복잡도에 따른 가중치를 결정하였다. 그러나 과거 프로젝트 데이터 수집을 통해 조직의 특성을 반영할 수 있는 단위 연산 규모 및 가중치를 활용할 경우 보다 높은 정확도를 기대할 수 있다.

### 4.2 트랜잭션 점수 계산 방법

조정 전 트랜잭션 점수(Unadjusted Transaction Point, UTP)는 액터의 가중치와 트랜잭션 내 연산의 가중치의 합으로 계산된다. 조정 전 트랜잭션 점수에 시스템 및 프로젝트의 특성을 반영하기 위해 조정 인자를 활용하여 트랜잭션 점수(Transaction Point, TP)를 계산하고 조직의 생산성(Person-Hour per TP)을 반영하여 공수를 예측한다. 구체적인 트랜잭션 점수의 계산 방법은 표 4와 같다. 음영으로 표시된 부분이 기존 유스케이스 점수 기법에서 변경된 단계이다.

## 5. 성능평가

### 5.1 실험 환경

표 2 트랜잭션 연산 단위 및 복잡도 테이블

| 연산 | 복잡도 | 분류기준 |
|---|---|---|
| 계산 연산 | 매우 낮음 | 간단한 계산 예)A=B+C*(D-E) |
| | 낮음 | 비교적 간단한 수준의 계산 예)D=SQRT(B**2-4*A*C) |
| | 보통 | 표준 함수 또는 기본적 행렬/벡터 연산 |
| | 높음 | 기본적인 수치분석 : 다항식 등 |
| | 매우 높음 | 복잡하고 구조적인 수치분석 : 비정칙 행렬 방정식, 간단한 병렬계산 |
| | 극히 높음 | 복잡하고 비구조적인 수치분석 : 높은 정확도를 요하는 노이즈 또는 확률적 데이터 분석, 복잡한 병렬계산 |
| 장치 의존적 연산 | 매우 낮음 | 간단한 형식을 통한 읽기 및 쓰기 연산 |
| | 낮음 | GET/PUT 수준의 입출력 연산 |
| | 보통 | 장치 선택, 상태 확인, 오류 프로세싱을 포함한 입출력 프로세싱 |
| | 높음 | 물리적 입출력 수준의 연산, 최적화된 입출력 오버랩 |
| | 매우 높음 | 인터럽트 식별, 서비스, 마스킹을 위한 루틴, 성능 중심의 임베디드 시스템 |
| | 극히 높음 | 마이크로 프로그램 연산, 성능이 매우 중요한 임배디드 시스템 |
| 데이터 관리 연산 | 매우 낮음 | 메인메모리 상의 배열 연산 |
| | 낮음 | 데이터 구조를 변화시키지 않고 데이터 갱신 없는 간단한 파일 부분 색인구축 |
| | 보통 | 여러개의 파일 입력에 하나의 파일 출력, 간단한 데이터 구조 변경, 복잡한 질의 및 갱신 |
| | 높음 | 복잡한 데이터 재구조화 |
| | 매우 높음 | 분산 데이터베이스 협조, 복잡한 트리거, 검색 최적화 |
| | 극히 높음 | 높은 커플링, 자연어 데이터 관리 |
| 사용자 인터페이스 관리 연산 | 매우 낮음 | 텍스트 기반의 입력 형식, 보고서 생성, 메시지 전송 |
| | 낮음 | 간단한 GUI 빌더 사용 |
| | 보통 | 간단한 위젯 사용 |
| | 높음 | 간단한 음성 입출력, 멀티미디어 입출력 |
| | 매우 높음 | 비교적 복잡한 2D/3D, 동적 그래픽스, 멀티미디어 |
| | 극히 높음 | 복잡한 멀티미디어, 가상현실, 자연어 인터페이스 |

표 3 단위 연산 복잡도에 따른 가중치

| 복잡도 | 매우 낮음 | 낮음 | 보통 | 높음 | 매우 높음 | 극도로 높음 |
|---|---|---|---|---|---|---|
| 가중치 | 0.73 | 0.87 | 1 | 1.17 | 1.34 | 1.74 |

실험에 사용된 총 9개의 시스템은 소프트웨어 공학 수업과정에서 학생들이 작성하였다. 이 시스템들은 약 3개월 간 개발되었고 Java 또는 C++을 통해 구현되었다.

이 실험을 통해 각 기법을 통해 얻어진 공수 예측 정확도를 서로 비교 분석한다. 각 기법의 정확도를 비교하기 위해 본 연구에서는 MRE(Magnitude of Relative Error)와 결정계수(R2)를 사용하였다. MRE는 값이 작을수록, $R^2$는 1에 가까울수록 기법의 정확도가 높다고 판단한다. 생산성 인자(productivity factor)는 실제 공수와 예측 공수 간 오차를 최소화할 수 있을 정도로 데이터가 축적된 경우를 가정하여 실제 생산성의 평균값을 사용하였다.

표 4 트랜잭션 점수 계산 절차

| 단계 | 규칙 | 결과 |
|---|---|---|
| 1 | · 액터의 분류에 따른 규모 계산 | 유스케이스 점수 기법과 동일 |
| 2 | · 유스케이스의 규모 계산<br>1) 트랜잭션 내의 각 연산별 개수 및 가중치를 통해 트랜잭션의 복잡도 계산 | Unadjusted Transaction Weights (UTW) = Σ(# of Unit operation * WF) |
| | 2) 트랜잭션의 복잡도를 합하여 유스케이스의 규모 계산 | Unadjusted Use Case Weights (UUCW) = ΣUTW |
| 3 | · 조정 전 트랜잭션 점수(UTP) 계산 | UTP = UAW + UUCW |
| 4 | · 조정 인자 계산 | 유스케이스 점수 기법과 동일 |
| 5 | · 조정 인자를 반영하여 트랜잭션 점수(TP) 계산 | TP = UTP * TCF * EF |
| 6 | · 생산성을 반영하여 공수 계산 | Effort = UCP * PHperTP |

### 5.2 성능평가 결과 및 분석

아래 표 5는 유스케이스 점수 및 트랜잭션 점수 기법을 적용한 예측 공수 및 MRE 결과이다. 실험 결과 총 9개 프로젝트 중 7개 프로젝트에서 트랜잭션 점수 기법의 MRE값이 유스케이스 점수 기법의 MRE 값보다 낮게 측정되었다.

t-test 결과 유의도는 0.010으로 95% 신뢰수준에서 트랜잭션 점수 기법의 정확도가 통계적으로 유의하게 유스케이스 점수 기법에 비해 향상되었음을 알 수 있었다. 또한 예측 모델의 설명력을 보여주는 $R^2$ 측정 결과

표 5 UCP 및 TP의 MRE(%) 결과

| 프로젝트 | UCP MRE(%) | TP MRE(%) |
|---|---|---|
| 1 | 13.87 | 9.64 |
| 2 | 9.67 | 11.09 |
| 3 | 0.47 | 2.78 |
| 4 | 22.07 | 0.27 |
| 5 | 51.13 | 31.06 |
| 6 | 34.14 | 11.56 |
| 7 | 19.45 | 12.91 |
| 8 | 23.38 | 14.09 |
| 9 | 39.18 | 22.05 |

역시 유스케이스 점수 기법은 7.01%에 불과한 반면 트랜잭션 점수 기법은 80.35%로서 트랜잭션 점수 기법이 좀 더 설명력 있는 기법임을 보여 주었다. 이는 공수 예측 시 트랜잭션 점수 기법이 비교적 안정적으로 정확도가 높은 결과를 기대할 수 있다는 점을 보여준다.

## 6. 결 론

본 논문은 유스케이스 모델을 확장하여 보다 정확한 공수를 예측할 수 있는 방법을 제안하였다. 기존 유스케이스 점수 기법에서 소프트웨어 규모 예측 단위로 사용하는 트랜잭션이 실제 균일한 복잡도를 가지지 않는다는 문제점에 착안하여 트랜잭션 내에 포함되어 있는 단위 연산을 유스케이스의 복잡도 측정 단위로 정의하고 이 단위연산의 복잡도를 통해 유스케이스의 규모를 측정하였다. 실험 결과 트랜잭션 점수가 기존 유스케이스가 모든 트랜잭션이 동일한 복잡도를 가진다고 가정한 문제점과 트랜잭션 개수에 따른 복잡도 상세화 문제를 해결하여 보다 정확한 공수가 가능함을 알 수 있었다.

본 논문에서는 네 가지 단위 연산이 개발 공수에 동일한 영향을 미친다는 가정을 기반으로 하였지만 실제 네 가지 단위 연산이 개발 공수에 미치는 영향은 상이할 수 있다. 따라서 향후 연구로서 하나의 조직에서 축적된 균일한 데이터를 획득하여 단위 연산별 가중치에 대한 분석을 수행할 예정이다.

## 참 고 문 헌

[1] B. Anda, H. Dreiem, D. Sjøberg and M. Jørgensen. Estimating software development effort based on use cases - Experiences from industry. In *Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools*, pp.487-502, 2001.

[2] B. Anda. Comparing effort estimates based on use cases with expert estimates. In *Proceedings of Empirical Assessment in Software Engineering*, pp.8-10, 2002.

[3] G. Karner. Resource estimation for Objectory projects. *Objectory systems*, 1993.

[4] J. Smith. The estimation of effort based on Use cases. *Rational Software*, White paper, 1999.

[5] B. Boehm, B. Clark, E. Horowitz, C. Westland, R. Madachy and R. Selby. Cost models for future software life cycle processes : COCOMO 2.0. *Annals of Software Engineering*, volume 1, number 1, 1995.

[6] C.R. Symons. *Software Sizing and Estimating MKII FPA (Function Point Analysis)*. Wiley-Interscience, 1991.

[7] http://www.ibm.com/developerworks/library/ar-usecases/

[8] A.J. Albrecht. Measuring application development productivity. In *Proceedings of the IBMApplic. Dev. Joint SHARE/GUIDE Symposium*, Monterey, pp.83-92, 1979.

[9] Mohagheghi, B. Anda and R. Conradi. Effort Estimation of Use Cases for Incremental Large-Scale Software Development. In *Proceedings of the 27th International Conference on Software Engineering*, pp.303-311, 2005.

[10] M. R. Braz and S.R. Vergilio. Software Effort Estimation Based on Use cases. In *Proceedings of the 30th Annual International Computer Software and Applications Conference*, vol.1, pp.221-228, 2006.