논문 2011-4-21

# 동시개발 소프트웨어 프로세스 모델

# Concurrent Software Development Process Model

최명복*, 이상운**

**Myeong-Bok Choi, Sang-Un Lee**

**요 약** 소프트웨어 프로젝트를 개발하는 방법론에는 20여 가지의 개발 프로세스 모델들이 존재한다. 그러나 모든 소프트웨어의 특성을 만족시킬 수 있는 일반화된 하나의 모델이 없는 실정으로 개발조직은 여러 모델들을 적절히 혼합하여 개발될 시스템과 개발팀의 능력에 맞도록 새로운 모델을 개발하여야만 한다. 본 논문에서는 다양한 소프트웨어 개발 상황에 보다 적합할 것으로 판단되는 동시개발 프로세스 모델을 제안한다. 먼저, 개발 요구사항 목록들이 작성되면, 요구사항을 중요도에 따라 20:80 비율로 분할하고, 중요한 20% 요구사항의 요구사항 분석과 아키텍쳐 설계가 완료될 때까지는 순차적으로 수행한다. 20%의 중요 요구사항에 대해 상세설계를 시작하는 시점에서 나머지 80%의 요구사항에 대한 요구사항 분석단계를 동시에 수행하는 개념이다. 동시개발은 타임박스(Timebox) 개념으로 수행되며, 이때 적용되는 순차적, 반복적 & 점진적 또는 Agile 방법들에 따라 각 타임박스에서 개발되는 요구사항의 분할 비율은 차이가 발생한다. 순차적, 반복적 & 점진적 또는 Agile 방법론을 동시개발 개념을 적용한 결과 단일화된 프로세스 모델로 표현할 수 있었다. 제안된 모델은 개발 단계들을 팀 단위로 수행할 경우 개발자원 활용의 비효율성을 크게 줄일 수 있다. 또한, 동시개발 개념을 적용하여 단계들이 중첩되어 수행되므로 개발기간도 크게 단축시키는 장점이 있다. 따라서 제안된 모델은 보다 빠른 시간에 보다 저렴한 비용으로 보다 좋은 품질의 소프트웨어를 개발하여 고객에게 납품할 수 있어 고객을 만족도를 향상시킬 수 있으며, 더불어 소프트웨어 개발 성공률을 높이는데도 기여할 것으로 판단된다.

**Abstract** Though a dozen of different software life cycle models are suggested, there is no universal model which can satisfy all the characteristics of software. Organizations mix and match different life cycle models to develop a model more tailored for their systems and capabilities. We suggest overlapped-concurrent development life cycle model that is more suitable in various software development environment. Firstly, we divided the development process into abstract and implementation stage. Abstract stage is from software concept phase to detailed design starting time, and implementation stage is from detailed design phase to system testing phase. Next, the abstract stage introduced the overlapped phase concept that begins the next phase when the step is completed 20% by applying pareto's law. In the implementation stage, we introduced the concurrent development which the several phases are performed some time as when one use-case (UC) is completed the next development phase is started immediately. The proposed model has an advantage that it can reduce the inefficiency of development resource greatly. This model can increase the customer satisfaction with a great product at a low cost and on a short schedule. Also, this model can contribute to increase the software development success rate.

**Key words :** Software Process Model, Timebox, Cascade Development, Concurrent Development, Pareto's Raw

*종신회원, 강릉원주대학교 멀티미디어공학과
**정회원, 강릉원주대학교, 멀티미디어공학과
접수일자 2011.3.13, 수정완료 2011.7.8
게재확정일자 2011.8.12

# Ⅰ. 서 론

고객은 일정 품질 수준 (Quality Level)을 만족하는 제품의 기능 (Scope)을 얻기 위해 개발자에게 시간 (Time)과 비용 (Cost)를 지불한다. 개발자는 주어진 시간과 비용 한도 내에서 고객이 원하는 품질수준을 만족시키는 제품 기능을 제공해야만 한다. 이를 프로젝트 4중 제약사항이라 한다.[1] 그러나 최근 들어 소프트웨어 기능은 보다 복잡해지고 고객은 보다 빠르고 적은 비용으로 소프트웨어를 획득하기를 원한다. 이러한 고객 요구사항을 만족시키기 위해 개발자는 소프트웨어 개발 프로세스 (또는 생명주기 모델)를 따른다.

소프트웨어 개발 프로세스는 그림 1과 같이 변화되고 있다. 1950~60년대는 단순한 소프트웨어 개발에 치중하여 프로세스를 전혀 적용하지 않아도 소프트웨어를 개발할 수 있었다. 따라서 이 시기에는 소프트웨어 개발 프로세스가 존재하지 않았다 (Nothing). 하드웨어 발전과 더불어 점차 복잡한 소프트웨어 시스템이 요구됨에 따라 소프트웨어를 보다 효율적으로 개발하기 위해 1970~90년대에는 엄격한 프로세스를 적용하였으며, 이는 엄격한 절차를 과다하게 요구하는 (Monumental 또는 Rigorous) 프로세스라 한다. 이후 보다 빠른 개발 요구를 만족시키기 위해 2000년대 초반부터는 민첩한 (Agile) 프로세스들이 등장하였다.[2]

소프트웨어 개발 프로세스에는 Pure Waterfall, V-Model, 2-Stage Waterfall, Sashimi, Waterfall with Subprojects, Staged Delivery, Design-to-Schedule, Waterfall with Risk Reduction, Spiral, Evolutionary Prototyping, Evolutionary Delivery Model, RUP (Rational Unified Process), Agile (XP, FDD, DSDM, SCRUM, AUP, …) 등 20여종이 존재한다. 그럼에도 불구하고 소프트웨어 개발 관리자는 제안서 작성 때나 실제 개발에 있어 최적의 개발 프로세스 선택이라는 어려움에 직면한다.

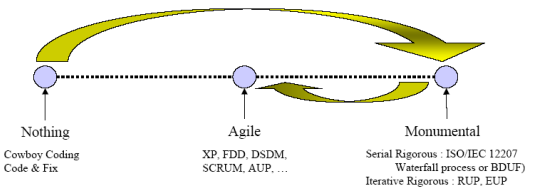



**그림 1. 소프트웨어 개발 프로세스 변천 과정**
Fig. 1. Change Process of S/W Development

소프트웨어 개발 프로세스들이 고객의 요구사항을 만족시키기 위해 발전되고 있지만 아직까지도 개발 성공률은 30% 이하이다.[3] 또한, 어떤 프로세스를 적용하는 것이 최적인가는 개발될 소프트웨어 특성에 의존한다. 비록 모든 프로젝트들이 가능한 빠른 개발이 요구되지만 다른 프로젝트는 다른 필요성을 가진다. 어떤 상황에서는 가장 빠른 생명주기모델이 다른 상황에서는 보다 느리기 때문에 모든 프로젝트 개발 상황에 적합한 하나의 프로세스는 없는 실정이다.[4-9] 예로, Jave.net[10]에서 883개 개발조직을 대상으로 소프트웨어 개발 프로세스를 적용하는 사례는 Nothing은 42.8%, Monumental Process는 21.6% (Waterfall: 11.2%, RUP: 10.4%), Agile Process는 27.4% (XP: 20.1%, SCRUM: 5.3%, DSDM: 2%), 기타 8%이다.

본 논문에서는 Monumental과 Agile을 모두 포함하는 단일화된 소프트웨어 개발 프로세스를 제안한다. 제안된 모델을 실제 적용하기 위해서는 Monumental이나 Agile 등 주어진 소프트웨어 개발에 가장 적합한 것으로 선택하여 동시개발 (Concurrent Development)하면 된다. Ⅱ장에서는 지금까지 제안된 생명주기 모델을 살펴보고 개발될 프로젝트 특성에 따른 모델의 성능을 살펴본다. Ⅲ장에서는 다양한 개발 환경에 가장 적합한 단일화된 프로세스 모델을 제안한다.

# Ⅱ. 소프트웨어 개발 프로세스 모델

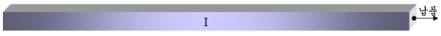
소프트웨어 개발 과정은 전형적으로 User's Software Requirements List, User Stories List나 Use Cases List 등을 결정하는 소프트웨어 개념정립 (SC, Software Concepts), 요구사항을 구체적으로 분석하고 모델링하는 요구사항 분석 (RA, Requirement Analysis), 시스템의 아키텍처를 설계하는 아키텍처 설계 (AD, Architecture Design, Preliminary Design 또는 High-level Design), 상세설계 (DD, Detail Design), 구현 (I, Implementation 또는 Coding), 단위시험 (UT, Unit Test), 통합시험 (IT, Integration Test), 시스템시험 (ST, System Test)과 수락시험 (AT, Acceptance Test)을 거친 후 고객에게 납품 (Delivery)된다.

소프트웨어 개발 과정에 대해서는 다양한 분류 방법이 존재하지만 본 논문에서는 SC, RA, AD, DD, I, UT,

IT, ST, AT로 통일하여 개발 프로세스들을 살펴본다.

그림 2는 Curren[4], Mohr[11], Dooley[7], Wallin과 Land[12], Kuhl[13], Huitsing[14], Cohen[15]과 Lewallen[16]를 비롯한 다양한 논문들로부터 획득된 자료들을 종합하여 현존하는 개발 프로세스들을 표현하고 분류한 것이다.

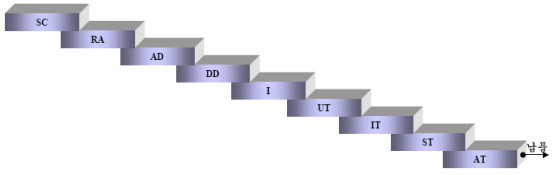
[Nothing]



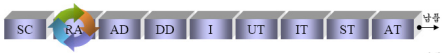

(a) Code & Fix (Build & Fix, Code-Like-Hell, Hacking)

[순차적 개발]


(b) Pure Waterfall Model (Basic Waterfall) & V-Model

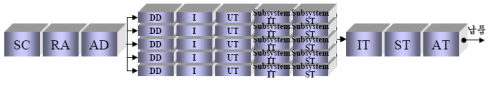



(c) Sashimi (waterfall model with overlapping phases, overlapped Waterfall) ≒ Salmon

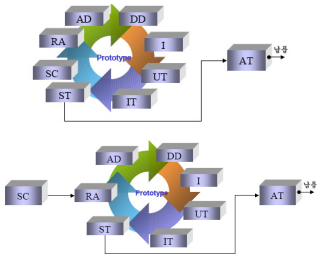



(d) Waterfall with risk reduction Model (Waterfall with spiral introduction, Spiral introduction to waterfall)

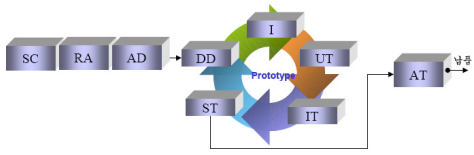
[반복적 & 점진적 개발]



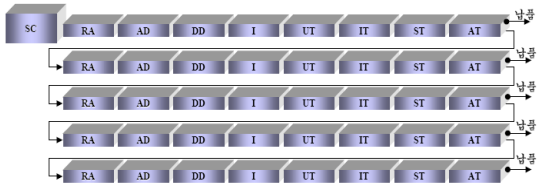

(e) Waterfall with Subprojects
(= parallel development model)



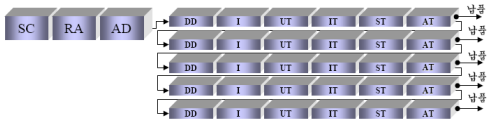

(f) RAD (g) Evolutionary Prototyping Model



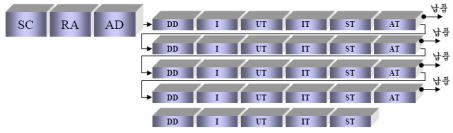

(h) Evolutionary Delivery Model



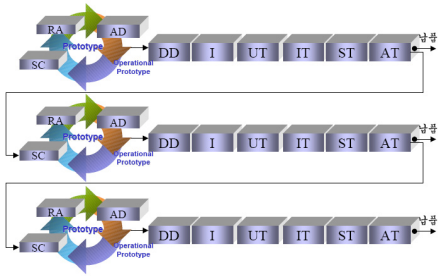

(i) RUP & Agile Process (XP, DSDM, FDD, …)




(j) Staged Delivery Model




(k) Design-to-Schedule Model




(l)Spiral Model

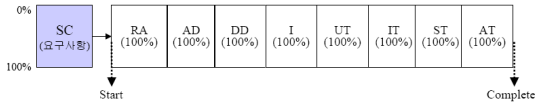
**그림 2. 소프트웨어 개발 프로세스 모델**
**Fig. 2. Process Model of S/W Development**

현존하는 개발 프로세스 모델들을 분류하는 방법에는 다양한 의견이 존재할 수 있으나 본 논문에서는 무(Nothing), 순차적 개발 (Sequential Development)과 반복적 & 점진적 개발 (Iterative & Incremental Development)로 분류한다. 이들 프로세스 모델들은 소프트웨어 개발 과정에서 프로젝트를 서브 프로젝트로 분할하여 개발하는 분할 비율 (양)의 차이, 어느 단계에서 분할하는가, 프로토타입을 만들어 점진적으로 구체화하는가의 차이가 있다.

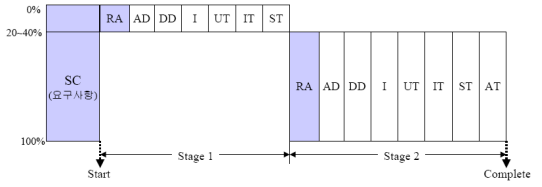
파이를 대상으로 이들 개발 방법론들을 설명하여 보

자. 순차적 개발은 SC 단계에서 개략적인 파이 형태를 형성하고 RA, AD, DD, I 등의 단계를 거치면서 파이 내부가 점차 명확해지는 형태이다. 반복적이고 점진적 개발은 프로토타입 개발 또는 서브시스템 분할 개발로 구분된다. 프로토타입 방법은 처음에는 파이 크기를 작게 구성하고, 점차 반복을 거치면서 파이의 크기가 증가함과 동시에 내부가 명확해지며, 최종적으로 완성된 파이를 얻는 형태이다. 서브시스템 분할 개발 방법은 파이의 크기를 분할하고 분할된 파이를 각각 구체적으로 개발하는 과정을 반복하면서 파이가 점진적으로 추가되며, 최종적으로 파이를 완성하는 방법이다.

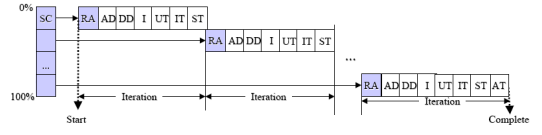
현존하는 개발 프로세스 모델들이 채택하고 있는 공통적인 개념은 단계나 반복이 순차적으로 수행되는 캐스케이드 (Cascade) 개발 개념 (한 단계 출력이 다음 단계의 입력에 연결)이다. 이들 공통 개념을 적용하여 순차적 개발과 반복적 & 점진적 개발 프로세스를 표현하면 그림 3과 같다.



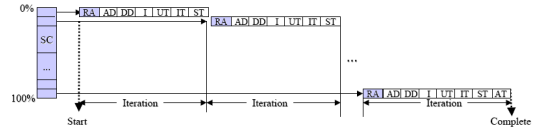

(a) 순차적(Pure Waterfall Model)




(b) 반복적 & 점진적 (2-Stage Waterfall Model)




(c) 반복적 & 점진적 프로세스 (RUP)




(d) 반복적 & 점진적 프로세스 (Agile Process)

**그림 3. 캐스케이드 (순차적) 개발 개념**
Fig. 3. Sequential Development Concept

폭포수 모델의 가장 큰 문제점은 BDUF 수행으로 요구사항 변경 (Requirements Creep)을 반영하지 못하는 점이며, RUP나 Agile 방법론은 서브시스템별로 점진적으로 납품하여 요구사항 변경을 수용하지만 시스템 아키텍쳐가 불안정한 상태에서 개발이 진행됨에 따라 아키텍쳐가 빈번하게 변경되어 리펙토링 (Refactoring)에 많은 어려움이 있을 수 있다. 또한, Agile은 터무니없이 많은 요구사항과 초기 반복에서 구현된 값비싸고 불필요한 장식 (Gold plating)으로 인해 끝이 보이지 않는 반복들이 수행되며, 주어진 예산을 모두 소진할 때만이 종료될 수 있는 단점이 있다.[17]

오늘날 복잡한 소프트웨어를 개발하기 위해서는 한 사람이 개발하기 보다는 일반적으로 팀을 구성하여 개발한다. 이러한 상황에서 어느 한 사람이 소프트웨어 개발 전 과정에 대한 전문적인 지식을 모두 갖추기는 현실적으로 어려우며, 대부분 어느 특정 과정에 대한 전문적인 지식만을 갖추고 있다. 이와 같이 특정 분야 전문지식을 갖춘 인원으로 팀을 구성하여 소프트웨어를 개발시 캐스케이드 방법론은 요구사항 분석, 아키텍쳐 설계, 상세설계, 코딩과 시험에 대해 다음 반복의 해당 단계까지의 기간 동안 수행하는 작업이 없어 인력과 시간 낭비를 초래한다.

이와 같은 문제점을 해결하기 위해 Ⅲ장에서는 시스템 아키텍쳐를 안정화 시킨 시점부터 시스템을 서브시스템으로 분할하여 동시에 정복하는 동시개발 프로세스 모델을 제안한다. 이와 같이 하면 요구사항 변경과 리펙토링 최소화를 시킬 수 있으며, 기존 프로세스 모델들 보다 개발기간도 단축시킬 수 있어 개발 성공률도 높일 수 있을 것이다.

## Ⅲ. 동시개발 소프트웨어 프로세스 모델

본 장에서는 그림 4와 같이 최적의 동시개발 시작 시점을 찾고, 시스템 아키텍쳐가 안정화 될 수 있는 최적의 요구사항 분할 비율을 구하여 시스템 아키텍쳐가 안정화된 시점부터 시스템을 서브시스템으로 분할하여 동시 개발하는 프로세스를 제안한다.

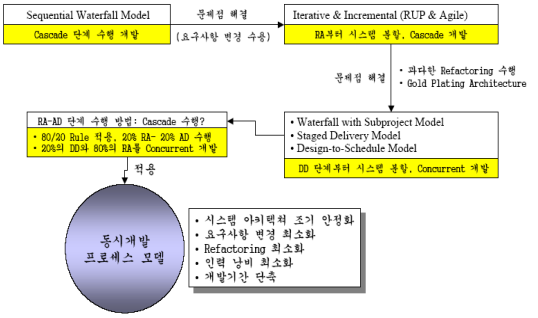
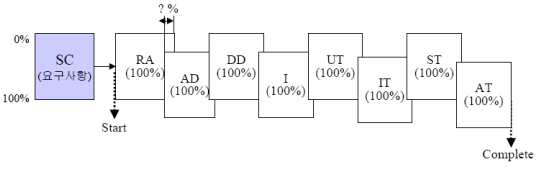
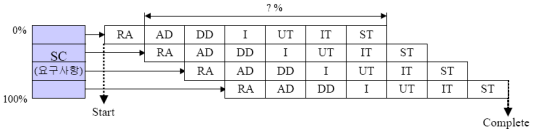



**그림 4. 동시개발 프로세스 유도**
**Fig. 4. Concurrent Development Process Derivation**

동시개발 프로세스를 유도하기 위해 먼저, 프로젝트 개발 일정을 보다 단축시킬 수 있는 방법으로 동시개발 개념을 적용한 프로세스 모델 연구가 있는지 살펴본다. 그림 5와 같이 폭포수 모델의 각 단계들을 중첩시켜 동시에 수행하는 프로세스는 Sashimi (중첩된 폭포수 모델)가 있다. 또한, RUP의 반복들을 중첩시켜 동시에 수행하는 방법에는 타임박스 프로세스 (Time Boxing Process)가 있다. 타임박스 프로세스는 반복적 & 점진적 프로세스와 관련된 개념으로 Jalote, Palit & Kurien[18]과 Crain[19]가 제안하고 있다. 따라서 프로젝트를 동시 개발하는 근거는 존재한다. 그러나 동시개발을 할 경우 보다 신중한 관리방법이 요구되며, 직면하는 문제점은 단계나 반복을 효율적으로 중첩시키는 양 (비율)이다. 그러나 각 단계나 반복의 중첩 비율이 설정되어 있지 않아 적용에 어려움이 있다.




(a) 중첩된 폭포수 모델 (Sashimi)




(b) 반복적 & 점진적 프로세스 (Timeboxing Process)

**그림 5. 동시개발 개념 적용 프로세스**
**Fig. 5. Concurrent Development Concept Process**

요구사항 변경과 리펙토링을 최소화 시킬 수 있도록 시스템 아키텍처 기반을 안정화 시키는 최적의 시점은 어느 단계인가? 먼저, 그림 2의 순차적 소프트웨어 개발 프로세스들을 고찰해 본 결과, SC-RA-AD 까지 수행하고 DD 단계부터 서브시스템으로 분할하는 방법이 최적이 될 수 있다. 이 방법에 속하는 모델들로는 Waterfall with Subprojects, Staged Delivery, Design-to-Schedule, Evolutionary Delivery와 Spiral이 있다. 따라서 본 논문에서는 DD 단계부터 시스템을 동시에 개발하는 방법을 채택한다.

다음으로 제기되는 문제는 SC 단계에서 얻은 요구사항 목록들을 분할하지 않고 RA-AD 단계를 순차적으로 1회만 수행할 것인가 아니면 분할하여 캐스케이드나 Concurrent로 수행할 것인가이다. 또한, 만약 요구사항을 분할한다면 분할하는 비율은 얼마로 할 것인가가 문제로 대두된다. 이러한 문제를 해결하기 위해 먼저, 소프트웨어 위기 (Software Crisis)를 해소하기 위한 방법들을 고찰해 보자.

개발된 소프트웨어가 예산 초과, 일정 지연, 낮은 품질, 고객 요구사항 불만족 결과를 나타내는 현상을 소프트웨어 위기라 하며, 1960년대 말에 Dijkstra[20]가 제기한 이후 지금까지도 해소되지 않고 있다. 소프트웨어 프로젝트 실패의 가장 근본적인 원인에 대해 Standish Group Report[21]는 "요구사항 불만족"에서 찾고 있다. 구체적으로는 사용자의 요구사항 제시 불충분 (13%), 불완전한 요구사항과 명세서 (12%)와 요구사항 변경 (12%) 등 프로젝트 실패 원인의 37%를 차지하고 있다. 소프트웨어 프로젝트가 성공하기 위해서는 요구사항 도출이 필수적이다. 소프트웨어 위기를 해소하기 위해 폭포수 모델에서는 과다하게 BDUF (Big Design Up Front)를 요구하게 되었으며, 요구사항 공학 분야도 연구되었다.[22] 다른 한편으로는 요구사항 변경을 수용하기 위해 서브시스템으로 분할하여 반복적으로 개발하는 RUP와 Agile도 제기되었다. RUP와 Agile은 RA 단계부터 요구사항을 분할하여 캐스케이드로 개발하기 때문에 Gold Plating 아키텍쳐를 유발시킨다. 또한, Agile은 설계를 전혀 하지 않는 것으로 착각하는 경우도 발생하여 BDUF가 조롱거리가 되거나 무시되어 디자인의 종말을 부르는 것 같다. 예로 XP는 Code & Fix로 회귀하기 쉽다.[23]

본 논문에서는 100% 요구사항 분석 후 개략설계를 수행하는 순차적 방법, 분할된 요구사항 각각에 대해 캐스

케이드로 요구사항 분석과 개략설계를 수행하는 반복적 & 전진적 방법과는 다른 방법을 제안한다. 이 방법은 20%의 중요한 요구사항 분석과 개략설계를 완료한 후 상세설계와 나머지 80%에 대한 요구사항 분석을 동시에 수행하는 방법을 채택한다. 이 방법을 제시하는 근거로 소프트웨어 개발 분야에서 파레토 법칙을 적용하고 있는 사례를 살펴보자.

파레토 법칙은 1900년도에 이탈리아 경제학자인 Vilfredo Pareto가 "일반적으로 20%의 사람이 80%의 부를 소유하고 있다"고 제안하였으며 80/20 법칙, Vital Few 법칙이라고도 한다.[24] 80/20 법칙은 놀랍게도 소프트웨어 개발 분야를 포함한 수많은 분야(소프트웨어 공학, 품질, 제조, 판매, 인력, 관리 등)에 적용되고 있다. 소프트웨어 분야와 관련된 80/20 법칙을 Kalaver[25], Schiller[26], Clair[27]을 비롯하여 다양한 자료들로부터 획득된 정보를 종합 정리하여 보면 "결과의 80%는 20%의 원인에 기인한다." "20%의 시험으로 시스템 고장의 80%를 검출한다." "사용자의 80%는 20%의 기능을 요구한다." "80%의 사업이익은 20%의 요구사항으로부터 얻는다." "시스템 기능의 80%는 20%의 요구사항에 기인한다." "20%의 노력으로 시스템 기능의 80%를 개발한다." "처음 일의 20%는 프로젝트의 80%를 생성한다." "20%의 작업자가 80%의 업무 가치를 창출한다." 등 다양하다.

전형적인 소프트웨어 개발 프로세스는 요구사항들로부터 시작된다. 요구사항에 대한 최적의 분할 비율을 고찰하여 보자. 80:20 법칙을 요구사항 관리에 적용하기 위해 "80%의 중요한 요구사항과 20%의 중요하지 않은 요구사항으로 분리한다."[28] "20%의 기능은 80%의 시스템 가치를 제공하기 때문에 개발 초기에 단지 20%의 가장 중요한 기능에 전력투구하여 조기에 성공을 달성하는 것이 필요하다."[29] 우리는 자원 (시간, 에너지, 사람, 돈)의 제약을 받기 때문에 한정된 자원으로 가장 큰 일을 먼저 수행하기를 원한다. "20%의 일은 당신에게 가장 큰 본전을 뽑을 수 있는 가치를 제공한다. 즉, 20%의 일로 80%의 결과를 얻을 수 있다. 이러한 20%를 찾아 귀를 기울여라."[30] 소프트웨어 공학 분야에서, 요구사항 변경 위험을 회피하고 올바른 기능에 관심을 집중시키기 위해서는 "적어도 80%의 사용자를 만족시키는 20%의 기능을 결정하는 것이 필요하다."[31] "원하는 결과의 80%를 생성할 수 있는 기능의 20%에 초점을 맞춘다."[32] "80/20 법칙을 사용하여 가장 큰 가치를 지닌 기능을 첫 번째로 납품하고 보다 비용이 요구되고 보다 중요하지 않은 기능들은 나중에 연구한다."[33]

Royce의 2-Stage Waterfall 모델[34]은 처음에는 완전히 개발하는 것이 없다고 가정하기 때문에 시스템 요구사항의 20~40%를 첫 번째 Stage에서 개발하고, 나머지 60~80%를 2번째 Stage에서 개발하는 방법을 택하고 있다. 이 개념은 RAD와 DSDM도 채택하고 있다. 또한, 2-Stage Waterfall 모델을 적용하여 RUP를 Agile로 개발하는 방법을 도입한 사례도 있다.[17]

RAD 방법론은 "가장 중요한 80%의 시스템 기능을 20%의 비용으로 먼저 개발하는 것이다."[35-38] DSDM에는 2가지 상반된 의견이 존재한다. 하나는 "80%의 사업이익은 20%의 설계 요구사항으로부터 나온다. 그러므로 DSDM은 사용자가 만족하는 충분한 20%의 기능을 처음으로 구현한다."[39]. 또 다른 의견은 DSDM은 "제안된 시스템의 유용한 80%의 기능을 20%의 노력으로 먼저 생성한다."[40]

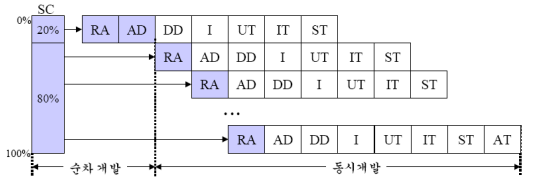
이와 같은 사례들로부터 SC 단계에서 파악된 초기의 100% 요구사항 목록에 대해 20:80으로 분할하여 중요한 20% 요구사항을 먼저 개발하는 것이 보다 효율적일 것이다.

따라서 본 논문에서는 "20%의 기능이 80%의 가치를 제공한다."는 80/20 법칙으로부터 "20%의 중요한 요구사항이 시스템 아키텍쳐의 80%를 결정한다."는 법칙을 유도하여 이를 적용하고자 한다. 20%의 중요한 요구사항으로부터 시스템 아키텍쳐의 기반이 결정되면 추후에 어떠한 기능이 추가되더라도 시스템 아키텍쳐가 그 것을 수용할 수 있다. 따라서 20%의 중요한 요구사항으로부터 도출된 시스템 아키텍쳐는 개발 초기에 안정화된 아키텍쳐를 얻을 수 있다.

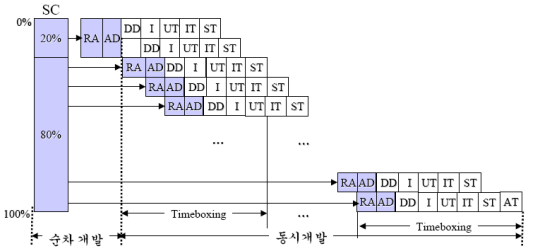
제안되는 프로세스 모델은 그림 6과 같이 SC 단계에서 소프트웨어 요구사항 목록이 작성되고, 최우선순위 요구사항 20%를 RA 단계에서 상세히 분석 완료하여 AD 단계에서 시스템 아키텍쳐 기반을 형성하고, 20%의 요구사항이 DD 단계를 수행하는 시점부터 나머지 80%의 요구사항에 대해 RA 단계부터 동시에 수행하는 방식이다. 이와 같이 동시개발 프로세스를 얻은 결과 순차적 엄격 (Sequential Rigorous), 반복적 & 점진적 엄격 (Iterative & Incremental Rigorous)과 반복적 & 점진적 민첩 (Iterative & Incremental Agile) 프로세스 모두 단일화된 프로세스로 통일시킬 수 있었으며, 단지 요구사

항 파이의 크기를 분할하는 양에만 차이가 있다.

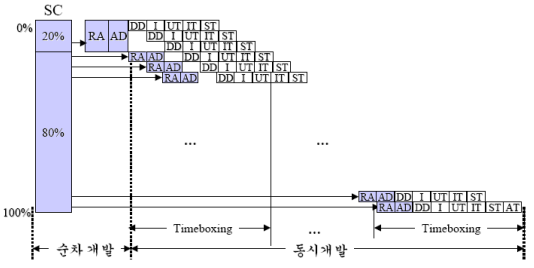
동시개발 프로세스의 SC, RA, AD와 DD 단계에는 ISO/IEC 12207, RUP, Agile 등 어떤 절차나 분석 모델링 기법들을 적용하여도 상관없다. 즉, 개발될 소프트웨어 특성에 따라 적합한 방법론을 채택하면 된다. 단지, 20%의 요구사항이 DD 단계를 수행하는 시점부터 동시개발을 수행하는 개념만 적용하면 된다.




(a) Sequential Rigorous Process




(b) Iterative & Incremental Rigorous Process




(c) Iterative & Incremental Agile Process

**그림 6. 동시 개발 프로세스 모델**
**Fig. 6. Concurrent Development Process Model**

동시개발 프로세스를 채택하면 시스템 아키텍쳐가 조기 안정화되고, 시스템의 중요 부분을 고객에게 조기 납품할 수 있으며, 동시 개발로 시스템 개발기간을 최대한으로 단축시킬 수 있다. 이로 인해 다음과 같은 효과를 얻을 수 있다.

(1) Agile 방법론에서 제기되는 리펙토링을 최소화 시킬 수 있다.
(2) 요구사항 변경을 최소화 시킬 수 있다.
(3) 개발기간 단축 및 인력 활용을 극대화로 인력 낭비를 최소화 시킬 수 있다.
(4) 빠른 납품으로 고객 만족도를 극대화 시킬 수 있다.

위의 효과 (3)은 그림 3과 그림 6을 비교하면 쉽게 알 수 있다. 그림 3의 반복적 & 점진적 RUP의 경우 하나의 서브시스템에 대한 RA 단계 (RA팀 수행) 종료와 다음 서브시스템 RA 시작 간에 시간 낭비가 많다. 반면에, 제안된 방법은 하나의 RA 단계를 완료한 이후 다음 서브시스템의 RA를 케스케이드 형태로 수행하는 방법으로 시간 낭비를 감소시킬 수 있으며, 이로 인해 전체 프로젝트의 개발일정을 크게 단축시킬 수 있다.

예를 들면 다음과 같다. RUP의 경우는 개발기간 동안 평균 8개 반복 수행, 만약, RUP(Rational Unified Process)로 12개월 프로젝트를 수행한다면 1개 반복은 1.5개월이 소요됨. 따라서 하나의 반복에서 요구사항 분석이 끝나고 다음 반복에서의 후속 요구사항분석이 시작될 때까지는 1.5개월의 공백이 있게 된다. 만약, 프로젝트를 8개의 서브프로젝트로 분할하여 하나의 서브프로젝트에 대한 요구사항 분석-개략설계-상세설계-코딩-시험이 1.5개월(45일)에 끝나고, 각각이 9일씩 걸린다면 제안된 방법은 요구사항 분석은 0일~72일(8*9), 개략설계는 10(9+1)일+72일, 상세설계는 19(2*9+1)일+72일, 코딩은 28일(3*19+1)+72일, 시험은 37일(4*9+1)+72일=109일 소요. 즉, 365일을 109일로 단축시킬 수 있게 된다.

## Ⅳ. 결론 및 향후 연구과제

소프트웨어 개발 과정은 특정한 단계들의 연속적인 수행으로 이루어진다. 이를 소프트웨어 개발 프로세스라 한다. 지금까지 다양한 개발 프로세스들이 제안되었지만 모든 소프트웨어의 특성을 만족시킬 수 있는 일반화된 프로세스가 없는 실정이다. 제안된 프로세스들은 개발 전 단계에 걸쳐 시스템의 기능을 전혀 분할하지 않고 개발하는 방법, 초기 단계부터 기능을 분할하여 개발하는 방법과 상세설계 단계부터 기능을 분할하여 개발하는 방법으로 분류할 수 있다. 이들 3가지 분류 모델들 각각은 장・단점을 갖고 있어 개발될 소프트웨어 시스템에 가장 적합한 하나의 프로세스를 선정하는데 어려움이 있다.

또한, 모든 프로세스들이 캐스케이드 개발 형태를 따르고 있어 개발 인력 낭비 뿐 아니라 개발기간도 단축시키기 어려운 단점이 있다.

본 논문에서는 다양한 소프트웨어 개발 상황에 보다 적합할 것으로 판단되는 동시개발 프로세스 모델을 제안하였다. 제안된 모델의 특징은 먼저, SC 단계에서 얻은 100% 요구사항 리스트를 기반으로 요구사항을 20:80으로 분할한다. 다음으로 20%의 중요한 요구사항에 대해 추상화 단계 (RA→AD)를 순차적으로 수행하여 시스템 아키텍쳐 기반을 형성하고 20% 요구사항에 대한 DD 단계를 시작하는 시점에서 나머지 80%의 요구사항에 대한 RA 단계를 동시에 수행하는 동시개발 개념을 적용하였다. 동시개발에는 한 단계에서 완료된 요구사항들이 바로 다음 단계 수행 팀에게 전달하여 여러 단계가 동시 수행되어 개발하는 형태를 취하고 있어 개발자원 활용의 비효율성을 크게 줄일 수 있는 장점이 있다. 또한, 여러 단계들이 중첩되어 동시에 수행되기 때문에 기존의 캐스케이드 개발 방법들 보다 개발기간을 현저하게 단축시키는 효과를 얻어 소프트웨어 개발 성공률을 높이는데도 기여할 것으로 판단된다.

아직까지 본 제안된 모델을 실제 적용하여 모델의 적합성을 검증하지는 못하였다. 따라서 추후 본 제안 모델을 실제로 적용하여 모델의 적합성 검증에 대한 연구를 수행할 예정이다.

## 참 고 문 헌


[1] J. Marasco, *"The Project Pyramid,"* The Rational Edge E-zine on-line magazine, http://www-128.ibm.com/developerworks/rational/library/4291.html, 2004.

[2] M. Fowler, *"The New Methodology,"* ThoughtWorks, 2005.

[3] B. Lewis, "The 70-Percent Failure," Infoworld. http://archive. infoworld.com/articles/op/xml/01/10/29/011029/opservival.html, 2003.

[4] K. Curran, *"Project Management: Project Lifecycle Plannin,"*http://www.infm.ulst.ac.uk/kevin/com820/week3.ppt, University of Ulster, Magee Collage, N. Ireland, 2005.

[5] T. Sloan, *"Development Models,"* http://www.nesc.ac.uk/talks/Wodsmes/day3/dev-models.pdf, National e-Science Centre, 2002.

[6] N. B. Priyanto, *"Software Project Management: Planning,"* Computer Science at University of Indonesia, 2005.

[7] J. F. Dooley, *"Life Cycle Models,"* http://deptorg.knox.edu/CS322/322PDFLectures/L2LifeCycle.pdf, KNOX College, 2005.

[8] D. F. Rico, *"Using Cost Benefit Analysis to Development Software Process Improvement (SPI) Strategies,* http://www.dacs.dtic.mil/techs/RICO/2-6cycles.shtml

[9] M. Lindvall and I. Rus, *"Process Diversity in Software Development,"* IEEE Software, Vol. 17, No. 4, pp. 14-18, 2000.

[10] Java.net, *"What Term Best Describes Your Software Development Process?,"* http://java.net/pub/pg/94, 1994.

[11] J. Mohr, "The Software Life Cycle," http://www.augustana.ab.ca/mohrj/courses/2005.fall/csc220/, Augustana Faculty, University of Alberta, Canada, 2005.

[12] C. Wallin and R. Land, *"Software Development Lifecycle Models: The Basic Types,"* Research Methodology for Computer Science and Engineering, 2001.

[13] J. J. Kuhl, *"Project Lifecycle Models: How They Differ and When to Use Them,"* Business eSolution, http://www.business-esolutions.com/islm.htm, 2002.

[14] S. Huitsing, *"Lifecycle Models,"* http://faculty.washington.edu/sharonh0/UW-HIA421/,UW Computing & Communications, 2001.

[15] S. Cohen, *"Software Life Cycles,"* Electrical & Computer Engineering, University of Pittsburgh, http://j7.ee.pitt.edu, 2005.

[16] R. Lewallen, *"Software Development Life Cycles Model,"* http://codebetter.com/blogs/raymond.lewallen/archive/2005/07/13/129114.aspx, 2005.

[17] M. Lines, J. Barnes, J. Holmes, and S. W. Ambler, *"Agile RUP: Experiences From The Trenches,"* http:// www.ibm.com/developerworks /library/edge/feb08/lines_barnes_holms_ambler/ index.html, The Rational Edge, 2008.

[18] P. Jalote, A. Palit, and P. Kurien, *"The Timeboxing Process Model for Iterative Software Development,"* http://www.cse.iitk. ac.in/jalote/papers/Time.boxingChap.pdf, 2005.

[19] A. Crain, *"Overlapping Iterations in a RUP -based Projects,"* http://www-106.ibm.com/ developerworks/rational/library/may05/crain/, The Rational Edge, 2005.

[20] E. Dijkstra, *"The Humble Programmer,"* Communications of the ACM, 1972.

[21] Standish group, *"The CHAOS Report,"* http://www.standishgroup.com/sample/PDFpages /chaos1994.pdf, 1994.

[22] E. Kazmierezak, *"Requirements Engineering, "* The University of Melbourne, 2003.

[23] M. Fowler, *"Is Design Dead?,"* Thoughtworks, 2004.

[24] B. Kaehms, *"80/20 Rule in Software Development,"* http://www.mnl.com/ourideas/opensource/8020 _rule_in_software_developm_1.php, Media Net Link, Inc., 2008.

[25] R. Kalaver, *"Prioritizing For Success; Living, Breathing 80-20,"*http://stealthemode.wordpress. com/2007/11/11/living-breathing-80-20/, wordpress.com, 2007.

[26] P. Schiller, *"The Critical, The Few, The Vital, But Not Necessarily The Only,"* http://www. leader2leaders.com/80-20, Leader2leaders, 2000.

[27] P. Clair, *"Evaluating and Managing Software Requirement Risk,"* Soutwest Research Institute, 2004.

[28] B. Kaehms, *"80/20 Rule in Software Development,"* Media Net Link, Inc., 2008.

[29] A. B. Pyster and R. H. Thayer, *"Software Engineering Project Management 20 Years Later,"* IEEE Computer Society, 2005.

[30] W. Bock, *"The Five Ps of Leadership,"* megastarmedia.com, 2003.

[31] Shmula.com, *"The Pareto Principle,"* http://www.shmula.com/129/the-pareto-principle, 2006.

[32] J. Zhuk, *"Integration-Ready Architecture and Design,"* Cambridge University Press, 2004.

[33] Envision Software, *"Methodology,"* http://www. envisionsoftware.com/MethodologyOverview. html, Envision Software, 2005.

[34] Royce, *"Managing the development of large software systems: concepts and techniques,"* Proceedings of the 9th international conference on Software Engineering, pp. 328-338, 1987.

[35] J. Martin, *"Rapid Application Development,"* Macmillan Coll Div, 1991.

[36] W. Maner, *"Rapid Application Development,"* http://www.cs.bgsu.edu/maner/domains/RAD.h tm, 1997.

[37] Xware.com, *"Putting the 'I' in 'IT',"* http://www.xware.com/files/credentials/CIO Advartorial_Bell_2006.pd, Xware.com, 2006.

[38] S. Adcock, *"Rapid Application Development: The Iterative Approach to Systems Analysis,"* http://www.umsl.edu/ ~asan74/, University of Missouri.

[39] Wikipedia, *"Dynamic Systems Development Method,"* http://en.wikipedia.org/wiki/Dynamic _Systems_Development_Method, Wikimedia Foundation, Inc., 2008.

[40] A. Irons, *"Agile Methods - Silver Bullet or Red Herring?,"* The British Computer Society, 2006.

## 저자 소개


### 최 명 복(종신회원)

- 1992년 : 호서대학교 전자계산학과(학사)
- 1994년 : 아주대학교 컴퓨터공학과(석사)
- 2001년 : 아주대학교 컴퓨터공학과(박사)
- 1997~현재 : 강릉원주대학교 멀티미디어공학과 교수
- 2004. 1~현재 : 한국인터넷방송통신학회 이사
- <주관심분야 : 지능형 정보검색, 퍼지응용, 지식표현, 신경망, 임베디드 및 유비쿼터스 응용, 지능형 교통제어, 소프트웨어 공학, 알고리즘>
- e-mail : cmb5859@gmail.com

### 이 상 운(정회원)

- 1983년~1987년 : 한국항공대학교 항공전자공학과 (학사)
- 1995년 ~ 1997년 : 경상대학교 컴퓨터과학과 (석사)
- 1998년 ~ 2001년 : 경상대학교 컴퓨터과학과 (박사)
- 2003년 : 강원도립대학 컴퓨터응용과 전임강사
- 2004년 ~ 2007.2 : 국립 원주대학 여성교양과 조교수
- 2007.3 ~ 현재 : 강릉원주대학교 과학기술대학 멀티미디어공학과 부교수
- <주관심분야 : 소프트웨어 프로젝트 관리, 소프트웨어 개발 방법론, 소프트웨어 척도 (소프트웨어 규모, 개발노력, 개발기간, 팀 규모), 분석과 설계 방법론, 소프트웨어 시험 및 품질보증, 소프트웨어 신뢰성, 신경망, 뉴로-퍼지, 그래프 알고리즘>
- e-mail : sulee@gwnu.ac.kr