학습이론을 이용한 소프트웨어 개발 프로세스 테일러링 기법

박수진, 나호영, 박수용, 양지훈

서강대학교 컴퓨터학과
서울 마포구 신수동 1 번지
{psjdream,hyna}@ selab.sogang.ac.kr, sypark@mail.sogang.ac.kr, jhyang@ccs.sogang.ac.kr

요약: 높은 소프트웨어의 품질은 유지하면서, 최소한의 비용으로 소프트웨어를 개발하기 위해서는 소프트웨어 개발 프로젝트의 상황에 알맞은 프로세스를 적용하는 것이 중요하다.일반적으로 상용 프로세스나 조직의 표준 프로세스를 프로젝트팀에 적용하고 있으나, 대부분의 경우, 경험부족이나 인력부족 등의 이유로 일반적인 프로세스를 어떤 가감도 없이 그대로 적용함으로써 오히려 소프트웨어 개발에 있어서 오버헤드를 초래하고 있다. 프로세스 테일러링 작업을 수행하는 경우에도, 대부분의 테일러링 작업은 몇몇 프로세스 엔지니어의 경험에 의존하는 실정이다. 이런 경우, 테일러링 결과로서의 프로세스는 얻을 수 있으나 타당한 근거를 제시하기 힘들고, 많은 시간을 요하게 된다. 따라서 본 논문에서는 인공신경망 기반의 학습이론을 프로세스 테일러링에 적용함으로써 테일러링 작업 중에서도 많은 시간을 필요로 하는 프로세스 필터링 작업을 자동화하는 방안을 소개하고 있다. 뿐 만 아니라 필터링 된 프로세스를 재구성하여 그 결과 얻어지는 프로젝트 상황에 적합하게 테일러링된 프로세스를 실제 프로젝트에 적용한 후 얻을 수 있는 피드백 자료를 학습의 자료로 다시 사용함으로써, 인공신경망의 정확도를 높여나가는 방법까지를 제시하고 있다. 본 논문에서는 이렇게 제시한 소프트웨어 개발 프로세스의 테일러링 방법의 실효성을 충분한 샘플자료를 바탕으로 한 실질적인 적용예를 통해 입증하고 있다.

핵심어: 프로세스 테일러링, 인공신경망, 프로세스 필터링, 태스크 컴포넌트, 프로세스 테일러링 파라미터

1. 연구배경

효과적인 소프트웨어 개발 프로세스를 정립하고 적용하는 것은 소프트웨어의 품질을 결정하는 가장 중요한 요소 중 하나이다. 효과적인 개발 프로세스

본 연구는 정보통신부 지원 ITRC 프로그램의 지원을 받아 수행되었음.

없이도 개발자의 노력에 따라 훌륭한 품질의 소프트웨어가 만들어질 수는 있으나, 고품질의 소프트웨어 개발이 한번으로 그치는 것이 아니라, 계속해서 지속되기 위해서는 체계적으로 정립된 개발 프로세스가 필요하다. 체계적인 소프트웨어 개발 프로세스와 프로세스의 지속적인 향상에 관련된 연구는 미국 SEI(Software Engineering Institute) 주도하에 만들어진 CMM(Capability Maturity Model)[4]관련 그룹이나, ISO 의 SPICE(Software Process Improvement and Capability dEtermination)[9]관련 그룹에 의해 활발히 진행되고 있다. 이런 모델들은 특정 프로젝트에 적용되고 있는 프로세스를 평가하고, 좀더 향상시킬 수 있는 방안들을 제시하고 있으나, 프로젝트 착수시에 새로운 프로세스를 구성하는 데 있어서 실질적인 가이드라인을 제시하지는 못하고 있다.

프로세스를 테일러링하는 작업은 현재까지는 프로세스 엔지니어들의 경험을 바탕으로 한 휴리스틱 방식에 의존해 왔다. 프로세스 엔지니어를 보유하고 있지 않거나, 프로세스 엔지니어의 프로세스 테일러링 경험이 충분하지 않은 조직에서, 상용 프로세스를 구입하여 적용할 경우에는 구입한 프로세스에 대한 충분한 이해가 없는 상태에서, 무조건적으로 적용할 수 밖에 없는 상황이었다. 경험이 풍부한 프로세스 엔지니어를 보유한 조직의 경우라도, 한두 명의 판단에 의지하여 1 개월 가량의 시간을 투자해야지만, 프로젝트팀의 상황에 맞게 테일러링된 소프트웨어 개발 프로세스를 얻을 수 있었다. 이렇게 얻어진 프로세스라 하더라도 기존의 테일러링 대상이 되었던 일반적 프로세스와 비교하여 누락된 태스크의 누락 이유와 참여하고 있는 태스크의 참여이유를 설명할 만한 체계적인 근거를 얻을 수는 없었다.

사람의 판단없이 프로세스를 테일러링하는 작업을 완전히 자동화하기란 불가능한 일이다. 그러나, 초보단계의 프로세스 엔지니어가 선배 엔지니어로부터의 간접경험을 통해, 혹은 자신이 실제 프로세스 테일러링 작업에 참여하면서 얻는 지식을 바탕으로, 새로운

프로세스에 참여하게 될 태스크들을 선별해 나가는 방법을 학습해 나가듯이, 시스템에게 동일한 방법으로 그 과정을 학습시키고, 지식을 끊임없이 확장시켜 나감으로써, 프로세스 테일러링 작업의 상당 부분을 자동화시켜 나가고자 하는 시도로부터 본 연구는 시작되었다.

본 논문에서는 특정 프로세스에 대해서 프로세스 엔지니어들이 보유하고 있는 테일러링 관련 지식을 재사용하거나, 혹은 그러한 지식을 기반으로, 새로운 프로젝트 환경이 주어졌을 경우, 그 환경에 적합한 형태의 프로세스를 재구성할 수 있도록 하는 매커니즘을 제시하고자 한다. 본 논문의 구성은, 먼저 2 장에서 관련 연구에 대해 설명하고 있다. 3 장에서는 프로세스 테일러링을 위해서는 일반적인 프로세스를 정의하는 프레임워크가 어떤 형태로 구축되어야 하며, 테일러링과 관련하여 축적된 전문가들의 지식이 어떤 형태의 기본적인 규칙으로 정의되며, 이렇게 정의된 규칙이 절대적일 수 없다는 점을 보완하기 위해, 학습기법이 어떤 식으로 응용되어 졌는지를 주축으로 하여, 전반적인 프로세스 테일러링 메커니즘을 설명하고 있다. 4 장에서는 기존의 휴리스틱 기법들과 비교하여 가장 차별화되고 있는 프로세스 필터링 단계를 실제로 적용한 예를 보여줌으로써 본 논문에서 제시하는 프로세스 테일러링 기법의 가능성을 보여주고 있으며, 마지막으로 5 장에서 본 연구의 결론을 맺으면서, 향후의 연구과제에 대해 언급하고 있다.

2. 관련연구

2.1 프로세스 테일러링(Process Tailoring)

프로세스 테일러링에 대한 연구는 일반 어플리케이션과 마찬가지로 소프트웨어 개발 프로세스 역시 모델링의 대상이 될 수 있다는 생각에서부터 1999 년 무렵부터 관련 논문들이 발표되고 있다. 그러나 그 이전에도 프로세스 엔지니어 그룹에서는 메소드 엔지니어링(Method Engineering) 분야로 꾸준히 연구되어져 왔다. 현재까지 이루어진 프로세스 테일러링 관련 연구는 프로세스 자체를 프로그래밍 하듯이 구조화하고 목적과 상황에 알맞은 형태로 변형시키기 위해 일관된 구조를 가지는 모델을 제시하고[13][14][15][16], 모델을 구성하는 요소들간의 관계를 규명하는데 초점이 맞추어져 왔다. 각각의 연구에서 각자의 모델 혹은 메타모델을 사용하여 테일러링 가능한 프로세스의 구조를 보여주고 있으나, 아래와 같은 공통점을 가지고 있다.

1. 일반적인 프로세스를 정의하는 부분과 프로젝트의 상황을 반영하기 위한 부분으로 나뉘어진다.
2. 프로세스 드라이버(프로세스의 형태를 결정하는 요소들) 역할을 하는 요소를 정의하고 있다.

위에서 프로세스를 나타내는 모델이 가지는 공통점 중 하나로 꼽은 프로세스 드라이버의 경우, 여러 가지 서로 다른 용어, 혹은 개념으로 설명되고 있으나 궁극적인 역할은 프로젝트의 특성을 반영하는 요소들을 말한다. 예를 들면, " 개발트랙(Development track)의 특성(characteristic)"[13], "상황요소(Contingency Factor)"[14],[15], "피쳐(Feature)"[16], "프로세스 식별자(Process Discriminant)"[3] 등을 들 수 있으며, 본 논문에서는 프로세스 테일러링 파라미터(Process Tailoring Paramter)로 정의되고 있다.

프로세스 테일러링이 동적으로 이뤄지기 위해서는 앞서 설명한 모델 위주의 접근방법만으로는 설명하기 힘들다. 따라서 잘 짜여진 프로세스의 구조 내에서 특정 프로젝트 상황이 주어졌을 때 해당 프로세스가 어떤 형태로 변화되어야 하는지에 대한 규칙이 필요하다. 현재까지는 프로세스 테일러링에 대한 선험적인 지식을 기반으로 하는 룰(rule)기반의 접근방법들이 소개되고 있다. 즉 아래 그림 1 과 같이 쌓여진 지식을 시작점으로 하는 추론방식을 이용하여 설명하거나 혹은 선언적인 가이드라인을 제시하는 방법이 주로 사용되고 있다.

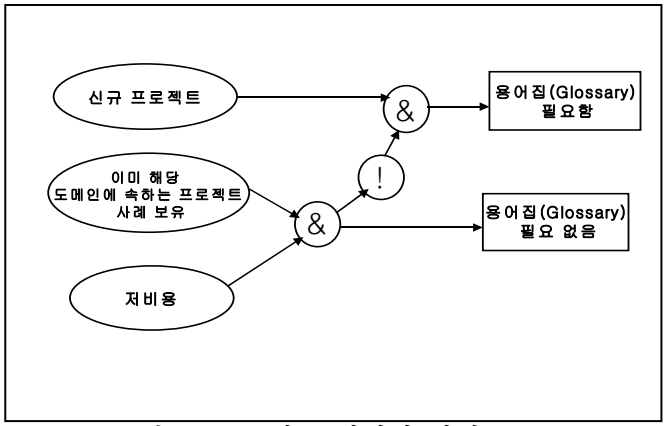

그림 1. 프로세스 지식의 단면[12]

이러한 추론 기반의 지식표현을 기반으로 하는 방법의 한계는 모든 발생 가능한 조건들을 미리 예상하여 정의해 둘 수 없다는 데 있다. 프로세스 드라이버(본 논문에서 지칭하는 프로세스 테일러링 파라미터)는 각각이 여러 개의 값들을 가질 수 있으며, 이 드라이버들 각각의 값이 만들어 내는 조합의 수는 수만 가지에 이른다. 또, 이 수만가지 조합에 대응되는 프로세스의 모양을 정의할 수 있다 하여도 그 내용은 시간이 흐름에 따라 불변하는 절대진리의 성격을 가지지 않으며, 기술적인 트렌드를 반영해 가면서 동일한 상황에서도 시간이 지남에 따라 상이한 결정

을 내릴 수 있다.

이와 같은 점에서 본 연구에서는 프로세스 테일러링 지식을 축적해 나가는 방법에 있어서 모든 지식을 정적인 형태의 규칙으로만 정의하는 것이 아니라, 끊임없이 진화하고 해당 조직의 환경에 적응해 나갈 수 있는 방식을 찾고자 하였고 그 대안으로 신경망 모델을 적용하기로 하였다.

2.2 태스크 컴포넌트(Task Component)[2]

소프트웨어 컴포넌트에 대해서 여러가지 정의가 있을 수 있으나 그 중 하나는 아래와 같다.

> 소프트웨어 컴호넌트란 계약적으로 명시된 인터페이스들과 명확한 문맥들 간의 의존관계만을 가지는 조합의 단위이다.[1]

이런 컴포넌트의 개념은 프로세스에 적용될 수 있다. 즉, 프로세스를 하나의 소프트웨어에 매핑하고 프로세스를 이루는 하부단위에 해당하는 액티비티들을 각각 프로세스라는 소프트웨어를 구성하는 각각의 컴포넌트로 볼 수 있다. 프로세스를 이루고 있는 가장 작은 작업단위인 태스크를 하나의 독립된 컴포넌트로 정의한다. 태스크 컴포넌트는 그 내부의 작업내용은 외부에 감추고 있으며, 외부에서는 이러한 태스크 컴포넌트를 하나의 프로세스가 되도록 조합하는 데 필요한 정보만을 볼 수 있을 뿐이다.

각각의 태스크 컴포넌트는 네 개의 인터페이스를 가진다. 각각의 태스크를 수행하기 위해 필요한 입력 산출물을 정의하는 입력 인터페이스와 태스크 수행결과 외부로 보내어지는 출력산출물을 정의하는 출력인터페이스, 해당 태스크가 특정 프로세스 구성에 포함되는지를 결정하는 데 필요 한 입력값들을 정의하고 있는 프로세스 테일러링 파라미터 인터페이스, 그리고 해당 태스크가 주어진 프로세스 테일러링 파라미터의 값과 무관하게 항상 수행되는 프로세스인지, 아닌지를 나타내는 옵션 인터페이스가 그것이다. 하나의 소프트웨어 개발 프로세스는 이런 구조를 가진 태스크 컴포넌트들의 조합이라고 볼 수 있다. 각 컴포넌트의 내부는 그림 2 에서 보이는 바와 같이 동일한 구조를 이루고 있음에 따라, 프로세스를 구성하는데 있어서 일관된 프레임워크는 유지하면서도 태스크의 다양한 조합을 통해 서로 다른 프로세스의 구성이 가능하다

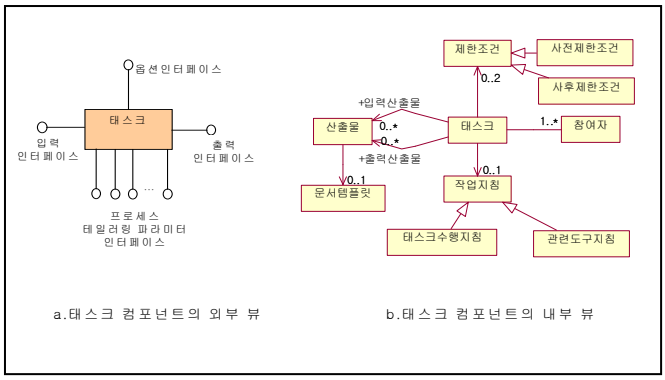

그림 2 태스크 컴포넌트의 a.외부뷰(view)와 b.내부뷰(view)

2.3 다층 전방향 신경망(Multi-Layered Feed Forward Neural Network [10]

다층 전방향 신경망은 입력층과 출력층 사이에 하나 이상의 중간층이 존재하는 신경망으로 그림 3 에 나타낸 것과 같은 계층구조를 갖는다. 이 때 입력층과 출력층 사이의 중간층을 은닉층 (hidden layer) 이라 부른다. 네트워크는 입력층, 은닉층, 출력층 방향으로 연결되어 있으며, 각 층내의 연결과 출력층에서 입력층으로의 직접적인 연결은 존재하지 않는 전방향(feedforward) 네트워크이다.

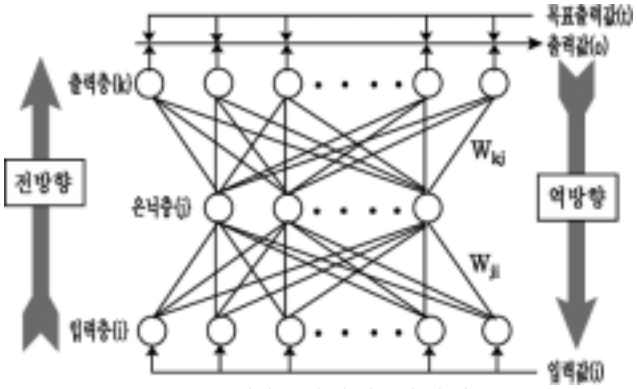

그림 3. 다층 전방향 신경망

다층 전방향 신경망은 단층 퍼셉트론(perceptron)과 유사한 구조를 가지고 있지만 중간층과 각 유닛의 입출력 특성을 비선형으로 함으로써 네트워크의 능력을 향상시켜 단층 퍼셉트론의 여러 가지 단점들을 극복했다. 다층 전방향 신경망은 층의 갯수가 증가할수록 퍼셉트론이 형성하는 결정 구역의 특성은 더욱 고급화된다. 즉 단층일 경우 패턴공간을 두 구역으로 나누어주고, 2 층인 경우 볼록산 (convex) 개구역 또는 오목한 폐구역을 형성하며, 3 층인 경우에는 이론상 어떠한 형태의 구역도 형성할 수 있다. 일반적인 다층 전방향 신경망의 학습방법은 다음과 같다. 입력

층의 각 유닛에 입력 데이터를 제시하면 이 신호는 각 유닛에서 변환되어 중간층에 전달되고 최종적으로 출력층으로 나오게 된다. 이 출력값과 원하는 출력값을 비교하여 그 차이를 감소시키는 방향으로 연결강도를 조정하는 것이다.

3. 프로세스 테일러링 기법 개요

프로세스 테일러링 기법은 세 단계로 나뉠 수 있다. 프로세스 테일러링 기법은 일반적인 프로세스로부터 프로젝트 상황을 고려하여, 채택될 태스크와 추출될 태스크를 구분하는 프로세스 필터링과 필터인(Filter-In) 태스크들의 선후관계를 설정하여 재구성하는 프로세스 재구성, 그리고 테일러링된 프로세스를 실 프로젝트에 적용한 결과를 다시 반영하는 프로세스 피드백 세 단계로 이루어져 있다. 아래 그림 4 는 이러한 프로세스 테일러링의 개요를 보여주고 있다.

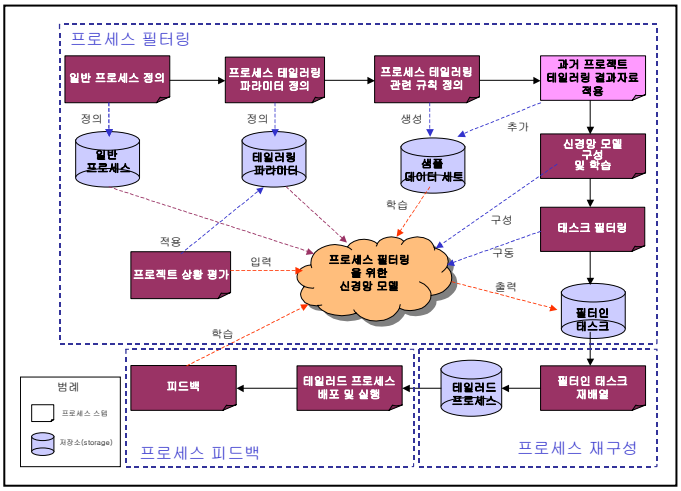

그림 4. 프로세스 테일러링 기법 개요

3.1 프로세스 필터링(Process Filtering) 단계

먼저 테일러링의 대상이 되는 일반 프로세스(generic process)를 선택해야 한다. 일반 프로세스는 해당 프로젝트가 속한 조직의 표준 프로세스일 수도 있고, 상용 프로젝트들 중에서 선택된 것일 수 있다. 일반 프로세스는 실제 프로젝트 적용시의 테일러링을 좀더 용이하게 할 수 있도록 정의되어야 한다. 프로세스는 태스크들로 이루어져 있으며, 각 태스크들은 하나의 정형화된 컴포넌트(2.2 참조)로 정의된다.

프로세스 필터링 작업은 다층 전방향 신경망 모델을 기반으로 한 학습과정을 통해 학습된 도구에 의해 자동으로 이루어질 수 있다. 프로세스 필터링에 앞서, 하나의 태스크가 특정 프로젝트에 적당한지를 판단하기 위해서는, 특정 프로젝트의 상황을 정확히 판단해야 한다. 정확한 프로젝트 상황 판단을 위해 프로젝트의 특징을 분별해 낼 수 있는 식별자가 필요한데, 이러한 식별자들을 프로세스 테일러링 파라미터(process tailoring parameter)로 사용한다. 프로세스 테일러링 파라미터는 조직의 특성에 따라 항목의 가감이 있을 수 있다. 프로젝트 관리자 혹은 프로젝트 팀원들을 대상으로 각각의 테일러링 파라미터에 대한 값을 결정하도록 하는 프로젝트 평가 워크샵을 수행하거나, 설문을 실행함으로써, 프로세스 테일러링에 쓰일 각 파라미터들의 값을 얻어낸다.

얻어진 파라미터들의 값은 다층 전방향 신경망의 입력으로 사용된다. 다층 전방향 신경망은 대량의 샘플데이터에 의해 이미 학습되어진 상태이다. 샘플데이터는 두 가지 방법으로 얻을 수 있다. 첫번째, 프로세스 테일러링에 대한 전문지식을 가지고 있는 프로세스 엔지니어들이 제시하는 제한조건(Constraint)과 테일러링 규칙(Tailoring Rule)을 기반으로 1 차 샘플데이터를 생성한다. 두번째, 이미 수행한 프로젝트의 경험을 나타내는 과거자료(historical data)가 있을 경우, 샘플데이터에 반영한다. 실제 프로젝트 경험을 바탕으로 한 데이터가 많을 경우, 좀더 해당 조직의 특징을 반영할 수 있는 형태로 신경망이 학습되어질 수 있다.

이 단계까지 수행하게 되면 프로젝트 개발팀은 아래와 같이 프로세스 필터링에 필요한 중간산출물들을 준비하게 된다.

1. 테일러링의 대상이 되는 일반 프로세스
2. 해당 프로젝트의 상황을 정의하고 있는 프로세스 테일러링 파라미터들의 실질적인 값
3. 샘플데이터 생성을 위한 기본적인 규칙과 규칙을 바탕으로 생성된 샘플데이터
4. 과거자료(historical data) : 특정 환경에 대비한 프로세스 테일러링의 결과
5. 3,4 를 기반으로 학습된 다층 전방향 신경망

위의 중간산출물들이 준비되면, 위의 목록중 2 번, 프로세스 테일러링 파라미터들의 값을 5 번, 학습된 신경망에 입력한다. 다층 전방향 신경망은 학습된 연결강도(weight)를 근거로 출력단의 각 태스크별 프로세스 적응도(adaptation degree)를 출력해 준다. 그림 5 는 이러한 일련의 프로세스 테일러링 작업의 모습을 보여주고 있다.

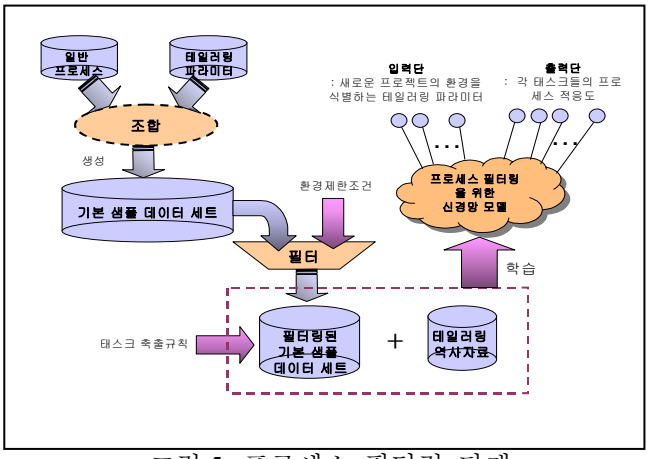

그림 5. 프로세스 필터링 단계

프로젝트 관리자는 출력된 각 태스크별 프로세스 적응도에 대한 최종적인 역치(Threshold)값을 지정할 수 있다. 예를 들어 역치값을 0.8 로 지정했을 경우, 0.8 이상의 프로세스 적응도를 보이는 태스크들만을 남기고 나머지 태스크들은 필터 아웃(filter-out)된다.

3.2 프로세스 재구성(Process Reconfiguration) 단계

프로세스 필터링 단계를 거쳐 선택된 태스크들은 2.2 에서 정의하는 바와 같은 태스크 컴포넌트 형태로 정의되어져 있다. 태스크 컴포넌트들은 입력산출물과 출력산출물을 인터페이스로 외부에 공개하고 있으므로, 이 두 가지를 기반으로 태스크간의 선후관계를 결정할 수 있다. 태스크간의 선후관계는 프로세스 필터링 단계의 “ 일반 프로세스 정의” 스텝 수행시에 이미 정의된다. 프로세스 재구성 단계에서는 미리 정의된 일반 프로세스를 이루고 있는 태스크간의 선후관계와 프로세스 필터링 단계를 거쳐 필터아웃(filter-out)된 태스트들의 선후관계로부터 필터인(filter-in) 프로세스들간의 선후관계를 설정해 준다. 필터아웃(filter-out)된 태스크 목록을 가지고 일반프로세스의 태스크 선후관계를 나타내는 테이블을 순차적으로 검색해 나가면서, 특정 테스크의 제외로 인해 끊어진 선후관계를 이어 나간다. 그림 6 는 이러한 과정을 자세히 나타내고 있다. 일반 프로세스에서도 모든 태스크들이 선후관계의 연결선 상에 존재하지 않듯이, 필터링을 거쳐 특정 프로젝트에 적용대상으로 선택된 태스크들 모두에 그 수행순서가 정해져 있진 않다. 예를 들어, “ 프로젝트 상황 모니터링” 같은 태스크들은 프로젝트 시작시점부터 종료시점까지 끊임없이 수행되어야 할 태스크로 다른 태스크와의 선후관계가 없다.

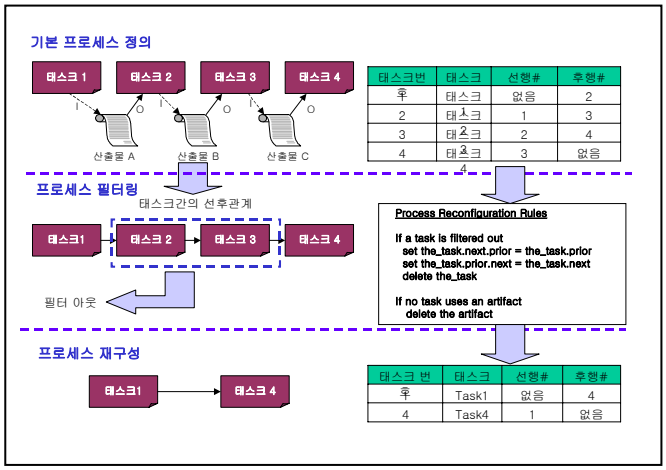

그림 6. 프로세스 재구성 단계

3.3 프로세스 피드백(Process Feedback) 단계

앞선 두 단계를 통해 특정 프로젝트에 알맞게 테일러링된 프로세스를 얻었다면, 그에 따라 프로젝트를 실행한다. 프로젝트 실행 결과 테일러링된 프로세스가 프로젝트 환경에 실제로 최적화된 프로세스였는지를 평가한다. 프로젝트에서 수행한 태스크들 각각의 유용한 정도를 1 부터 5 까지 다섯 단계로 나누어 태스크를 수행한 프로젝트 팀원에게 측정하도록 한다. 설문의 형태로 측정할 수도 있으며, 상황이 허락할 경우, 1 일 정도의 워크샵을 통해 측정할 수도 있다.

태스크의 적절성에 대한 평가가 완료되면, 각 태스크별 적절성이 4 이상인 것은 ‘ 적용’ 으로, 3 이하인 것은 ‘ 미적용’ 으로 표시한 후, 앞서 프로세스 필터링 단계 중 프로젝트 평가를 통해 얻어진 프로세스 테일러링 파라미터와 함께 한 건의 샘플데이터로 신경망 모델에 반영하여, 신경망 모델의 정확도를 높여나가도록 한다. 프로세스 피드백을 통해 하나의 조직에서 쌓여지는 프로세스 테일러링 관련 지식은 횟수를 거듭할수록 확장되어 간다. 즉, 앞선 프로세스 테일러링 관련 과거자료(historical data)와 실제 테일러된 프로세스의 적용결과는 뒤따라 발생하는 프로세스 테일러링 작업시에 재사용되는 효과가 발생한다.

4. 프로세스 필터링 적용 예

본 논문에서 제시하고 있는 프로세스 테일러링 기법 중 기존의 휴리스틱한 방법들과 비교하여 가장 새로운 접근이라고 할 수 있는 프로세스 필터링 단계의 이해를 돕기 위해 업계에서 사용되고 있는 상

용 프로세스를 테일러링 대상 프로세스로 하여 실제 적용해 보았다. 다음은 프로세스 필터링 단계를 이루고 있는 각 스텝을 실제 수행하는 데 있어서의 자세한 기법 설명과 각 스텝별 중간 산출물들을 보이고 있다.

스텝 1. 일반 프로세스 정의

본 논문에서 제시하는 프로세스 테일러링 기법을 적용하기 위해, 테일러링의 대상이되는 일반 프로세스는 태스크 컴포넌트들의 조합 형태로 전환되어야 한다. 일반프로세스로 특정 조직의 표준 프로세스를 적용할 경우, 데이터 샘플 수집의 한계가 있다는 점을 고려하여, 현재 소프트웨어 개발 프로세스로 광범위하게 사용되고 있는 상용 프로세스인 RUP(Rational Unified Process)[3]를 일반프로세스로 채택하였다. RUP 는 9 개의 디스플린(discipline)으로 이루어져 있으며, 각각의 디스플린은 액티비티들(activity)의 시퀀스(sequence)로 이뤄진 워크플로우(workflow)들로 구성되어 있다. 이 중에서 태스크 컴포넌트에서 정의하는 입출력 산출물을 명시하고 있는 작업단위는 액티비티이므로, 각각의 액티비티들을 태스크 컴포넌트로 매핑하였다. 각 태스크의 선후관계는 각 태스크가 외부로 공개하고 있는 입출력 인터페이스에 명시된 산출물들간의 관계를 고려하여 부여되었다. RUP 는 상용 프로세스인 만큼 포함하고 있는 액티비티의 숫자도 총 148 개에 이른다. 본 실험에서는 이 148 개의 액티비티 중에서 프로젝트 상황에 대해 그 적용여부의 결정이 민감한 프로젝트 관리 디스플린(Project Management Discipline)에 속하는 액티비티들을 프로세스의 테일러링 대상이 되는 일반 프로세스를 구성하는 태스크 컴포넌트로 최종 결정하였다.

스텝 2. 프로세스 테일러링 파라미터(Process Tailoring Parameter) 정의

프로젝트의 상황을 정확히 정의하기 위해서는 일정한 평가항목(metric)들이 필요하다. 현재까지 제시된 소프트웨어 관련 평가항목들은 소프트웨어의 품질에 관한 것이거나[6], 소프트웨어 프로세스 자체를 평가하기 위한 것[4]이었다. 본 연구에서는 RUP[3]의 프로세스 식별자(Process Discriminants)와 Walker Royce 의 Software Project Management[7]를 바탕으로 하여, 프로세스 필터링의 드라이버(driver) 역할을 하는 프로세스 테일러링 파라미터(Process Tailoring Paramter 를) 표 1 과 같이 정리하였다. 본 논문에서 제시하고 있는 프로세스 테일러링 파라미터 외에도 조직에 따라 다른 항목을 추가하거나 삭제할 수 있다. 그러나 반드시 특정 태스크의 프로세스 참여 여부를 결정하는데 명확한 식별자 역할을 할 수 있어야 하며, 노미널(nominal)형태로 그 값을 판단할 수 있어야 한다.

항목	노미널 값(nominal value)	숫자값
외부요소		
프로젝트 개발형태	계약기반의 발주 프로젝트	4
	상업적인 제품개발 프로젝트	3
	내부 프로젝트	2
	실험적 프로젝트	1
프로젝트 관련자 협력관계	높음	1
	낮음	2
내부요소		
프로젝트규모	매우 큼 (Over 125 people)	4
	큼(26 people~ 125 people)	3
	보통(6 people ~ 25 people)	2
	적음(~ 5 people)	1
지리적 분포	분산	2
	집중	1
개발팀의 프로세스 성숙	높음	1
	중간	2
	낮음	3
개발팀의 도메인 경험	높음	1
	중간	2
	낮음	3
대상시스템의 특징		
새로운 정도	높음	3
	중간	2
	낮음	1
아키텍처 측면의 위험요소	높음	3
	중간	2
	낮음	1
기술측면의 복잡도	높음	3
	중간	2
	낮음	1
관리측면의 복잡도	높음	3
	중간	2
	낮음	1

표 1. 프로세스 테일러링 파라미터

프로세스 테일러링 파라미터는 신경망 모델 구성시에 입력속성(input attribute)으로 매핑되며, 프로세스 테일러링 파라미터에 할당되는 값은 입력속성의 값으로 매핑된다. 신경망 모델의 입력속성값으로 사용되기 위해 프로세스 테일러링 파라미터 값은 숫자형태로 전환되어야 한다. 프로세스 테일러링 파타미터의 노미널 값을 숫자로 전환할 때에는 프로세스의 정형성을 높이는 쪽으로 갈수록 큰 값을 할당한다. 예를 들어, 관리측면의 복잡도가 높으면 높을수록 프로세스의 정형성 역시 높아진다고 볼 수 있다. 따라서 관리측면의 복잡도가 " 높음" 일 경우엔 3, " 중간" 일 경우엔 2, 마지막으로 " 낮음" 일 경우 3 으로 전환된다.

스텝 3. 프로세스 테일러링 관련 규칙 정의

본 연구에서 가장 어려웠던 점은, 프로세스 테일러링 파라미터들의 조합이 생성해 내는 프로젝트 환경의 수는 46000 여개에 이르는 반면, 실제 프로세스로부터 얻어지는 결과에 해당되는 샘플데이터의 수집 숫자는 너무 작다는 것이었다. 국내 소프트웨어 개발 관련 업체들 중, CMM[4] 레벨 4 이상을 획득하여, 프로젝트의 모든 상황을 수치화하고 그 수치들을 평가해 나가고 있는 업체는 4~5 개 정도이며, 이런 업체에서 이미 수행한 과거자료(historical data)를 모두 수집할 수 있다고 가정하여도, 1000 개를 초과하기 어렵다. 이런 한계점을 극복하기 위하여, 아래와 같은 방법으로 접근하였다.

프로세스 테일러링 파라미터들의 모든 가능한 조합 수만큼의 샘플데이터를 자동 생성하고 모든 태스크의 프로세스 적절성을 디폴트 값 1 로 설정한다. 환경 제한조건(environment constratint)를 사용하여 프로세스 파라미터들간의 관계에서 서로 일관되지 않은 값을 포함하고 있는 샘플데이터를 데이터 셋에서 제외시킨다. 또한 이미 프로세스 엔지니어가 보유하고 있는 프로세스 테일러링 관련한 선험적인 지식을 태스크 축출규칙(task subtraction rule)으로 정의하여, 샘플데이터에 적용함으로써, 입력된 프로세스 파라미터들 값의 특정한 한 세트, 즉 특정 프로젝트 상황에 대한 개개의 태스크들의 프로세스 적절성을 0 으로 설정해 나간다.

위의 접근방법에서는 두 가지 형태의 프로세스 테일러링 규칙을 정의하여 자동생성된 샘플데이터 세트에 적용하고 있다. 첫번째가 프로세스 테일러링 파라미터들간의 관계를 바탕으로 한 환경 제한조건(environment constratint)이다. 예를 들면, 프로젝트 개발형태가 " 상업적인 제품개발 프로젝트" 인 경우, 불특정 다수의 미래 고객을 대상으로 하고 있으므로 프로젝트팀 외부의 스택홀더(Stakeholder)와의 협력관계는 높을 수 없다. 따라서 " IF Type of Development = 3 THEN Stakeholder cohesion or contention = 2" 라는 환경 제한조건을 추출할 수 있고, 이 규칙에 의해 프로젝트 개발형태의 값이 3 이면서 프로젝트 관련자 협력관계의 값이 2 가 아닌 샘플들은 샘플데이터 세트에서 제외시킬 수 있다. 이런 환경 제한조건에 따라, 46000 여개의 모든 프로젝트 환경을 나타내는 파라미터조합 중에서 20000 여개 정도는 제외되고 나머지 26000 여개가 전체 샘플데이터 세트의 크기가 되었다..

두 번째로 샘플데이터 세트에 대해 이미 알고 있는 프로세스 테일러링 관련 지식들을 규칙으로 정의할 수 있는데, 이것을 태스크축출 규칙(Task Subtraction Rule)이라 한다. 예를 들어, RUP 의 프로젝트 관리 디스플린에 속하는 액티비티 중에서 " 비즈니스 유스케이스 작성" 의 경우, 실험적 프로젝트이거나 프로젝트의 규모가 적을 경우, 적용하지 않는다는 규칙이 있다면 아래와 같이 정의될 수 있다.

IF (Type of Development = 1) OR
(Project Scale = 1)
THEN Task Appropriateness of Develop Business Use Case = 0

위의 규칙을 샘플데이터 세트에 적용할 경우, 조건부와 일치하는 프로세스 테일러링 파라미터 값을 가지는 샘플데이터의 해당 태스크에 대한 태스크별 프로세스 적응도(adaptation degree)는 0 으로 설정된다. 이처럼, 선험적인 지식이 반영된 샘플데이터를 신경망에 학습시킴으로써 프로세스 엔지니어 개개인이 보유하고 있던 지식이 간접적으로 신경망에 축적되는 효과를 얻을 수 있다.

스텝 4. 과거 프로젝트 테일러링 결과자료 적용

앞 절에서와 같이 특정 태스크의 프로세스 적응도(adaptation degree)를 출력으로 하는 신경망을 학습시키기 위한 샘플데이터 세트는 기존의 사전 지식을 기반으로 만든 규칙에 따라 생성할 수 있다. 여기에 신경망의 정확도를 높이기 위해 이전에 프로세스 테일러링 작업시 남겨놓은 과거자료(historical data)가 있을 경우, 샘플데이터 세트에 추가 반영한다. 즉, 프로세스가 처한 특정 환경과 주어진 환경에 맞게 테일러링 된 프로세스에 선택된 태스크들 쌍이 한 건의 샘플 데이터로 반영한다. 이런 보정 작업을 통해, 특정 조직이 지금까지 수행해 온 테일러링의 조직적 특성이 반영될 수 있다. 이 스텝은 해당 조직의 과거자료(historical data)가 없을 경우는 생략한다. 본 연구에서는 RUP 를 일반프로세스로 선택한 만큼, 한국

IBM Rational 사의 프로세스 엔지니어로부터 이러한 프로세스 테일러링 관련(historical data)를 수집하였다. 수집된 샘플데이터의 개수는 368 개로 그 중 37 개의 데이터는 테스트 데이터로 사용되었고 나머지는 학습데이터로 사용되었다.

스텝 5. 태스크의 프로세스 적응도 산출을 위한 신경망 모델 구성 및 학습

본 적용예에서는 태스크의 프로세스 적응도 산출을 위한 신경망 모델 구성을 위해, Java-NNS(Java-Neural Network Simulator)[5]라는 신경망 모델 시뮬레이터를 사용하였다. 프로세스 테일러링 파라미터를 입력으로 받아, 각 태스크들의 프로세스 적응도를 출력하는 3 층 순방향 신경망(3-Layered Feed Forward Neural Network)을 구성하였는데, 이는 입력으로 사용되는 파라미터값들과 출력으로 산출되는 각 태스크의 프로세스 적응도 간의 명확한 관계를 파악하고 있지 않은 상태에서 학습시키기에 적합한 모델이기 때문이다. 학습 알고리즘으로는 백프로퍼게이션(backpropagation)을 사용하였다. 은닉층(hidden layer)의 노드(node) 숫자는 네트워크 설계에서 많은 이슈가 되는 부분이나, 널리 쓰이고 있는 규칙 중 하나인 " 은닉층의 노드 개수 = (입력노드의 개수+ 출력노드의 개수) * 2/3" [11]에 따라서 (10+30)*2/3=27 개의 노드를 은닉층에 할당하였다. 이렇게 생성된 신경망 모델의 모습은 그림 7 과 같다.

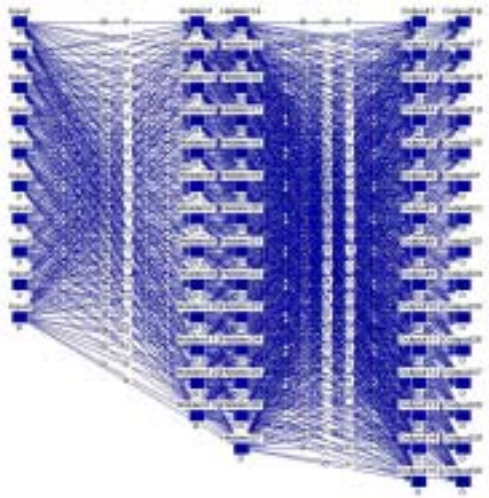

그림 7. 프로세스 적응도 산출을 위한 신경망 모델

그림 7 과 같이 구성된 신경망 모델을 학습시키기 위해 스텝 3,4 를 통해 얻은 샘플데이터를 학습데이터로 사용하였다. 학습데이터의 구성과 신경망의 정확도 사이의 상관관계를 알아보기 위해 표 2 와 같이 4 개의 그룹으로 나누어 학습시켰다. 공정한 정확도 비교를 위해 테스트 데이터는 모두 동일한 샘플 데이터를 사용하으며, 과거자료의 양이 신경망 모델의 정확도에 미치는 영향을 살펴보기 위해 순차적으로 학습데이터에 반영하는 과거자료의 양을 늘려 나가면서 신경망 모델의 정확도를 관찰해 보았다.

그룹	학습데이터	테스트 데이터
그룹 1	스텝 3 의 규칙을 통해 생성된 기본 샘플 데이터	스텝 4 를 통해 수집한 과거자료 중 37 개의 샘플데이터
그룹 2	스텝 3 의 규칙을 통해 생성된 기본 샘플 데이터 + 스텝 4 를 통해 수집한 110 개의 샘플 데이터	
그룹 3	스텝 3 의 규칙을 통해 생성된 기본 샘플 데이터 + 스텝 4 를 통해 수집한 220 개의 샘플 데이터	
그룹 4	스텝 3 의 규칙을 통해 생성된 기본 샘플 데이터 + 스텝 4 를 통해 수집한 331 개의 샘플 데이터	

표 2. 신경망 모델 테스트를 위한 샘플데이터 구성

4 개의 학습된 신경망 모델에 동일한 37 개의 테스트 데이터를 입력하여 출력된 각 태스크의 프로세스 적응도(1 또는 0 으로 판단함)와 미리 프로세스 엔지니어에 의해 판단된 각 태스크의 프로세스 적응도 사이의 오차를 나타내는 에러율의 추이는 아래 그림 8 과 같다. 8 번 샘플데이터에서 다른 데이터와는 달리 상당히 큰 에러가 발생하는 것을 볼 수 있는데, 이는 테스트 샘플을 작성한 프로세스 엔지니어의 견해가 신경망 모델을 학습시킬 때 사용된 학습데이터에 해당되는 다른 샘플데이터와 큰 차이가 있었음을 의미한다. 그 외에 샘플데이터에서는 최대 30% 미만의 에러를 보이고 있으며, 37 개의 테스트 결과의 평균에러율은 그림 9 과 같이, 약 16% 내외이다.

평균에러율의 변화를 보면, 스텝 3 을 통해 얻어진 규칙 기반의 기본 샘플 데이터만을 학습시킨 경우 가장 낮은 평균에러율(15.8%)을 보이다가 과거자료에 해당되는 샘플데이터를 처음 추가했을 때 증가(17%)한 후, 다시 순차적으로 과거자료의 양이 증가될수록 그 에러율이 감소되고 있다. 이는 규칙기반의 기본 샘플 데이터만을 학습시킨 경우, 샘플데이터들은 프로세스 엔지니어들이 정의한 균일한 규칙을 반영하고 있는 형태이므로, 신경망 모델에 학습된 상태는 상당히 안정적이라고 할 수 있다. 따라서, 일반적인 대전제에 해당하는 규칙과 프로세스 엔지니어들의 개인적인 판단견해와 비교하는 결과가 되므로 가장 작은 편차를 나타내고 있다. 이에 반해 균일한 규칙

의 적용을 받는 데이터 세트에 특정 프로젝트의 프로세스 테일러링 결과를 처음 적용할 경우, 이 샘플 데이터들은 신경망 모델의 관점에서는 일종의 노이즈(noise)로 작용함에 따라 일시적으로 평균에러율이 증가하게 된다. 그러나, 과거자료들이 계속해서 학습 데이터에 추가됨으로써 다시 평균에러율은 감소추세로 돌아서고 있음을 보이는데, 이는 신경망 모델이 해당 조직의 프로세스 테일러링 특성이나 추세를 반영하는 형태로 적응해 나가고 있음을 의미한다.

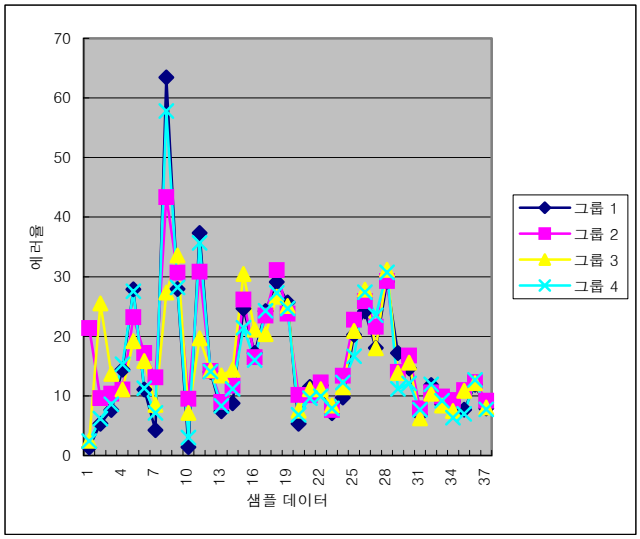

그림 8. 샘플데이터별 에러율 비교

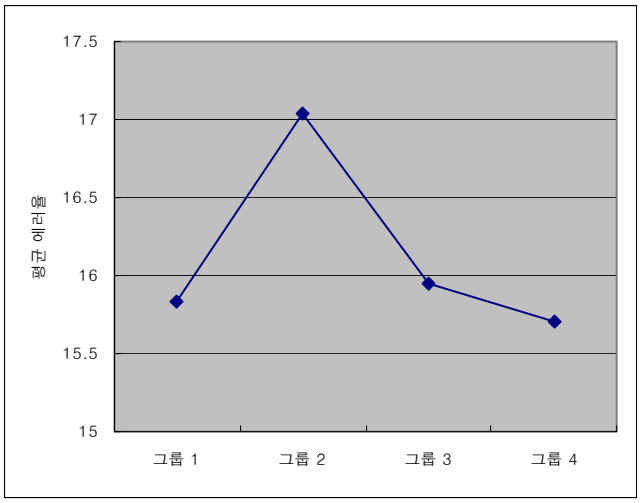

그림 9. 평균 테스트에러율 비교

본 스텝에서는 원래의 역할인 신경망 모델 구성과 학습에 덧붙여, 신경망 모델을 프로세스 필터링에 사용하는 방법의 실효성을 보이기 위해 몇가지 실험을 해 보았다. 그 결과, 신경망 모델의 신뢰도가 평균 83%를 상회하고 있으며, 과거자료의 양이 계속해서 학습에 반영될수록 해당 조직의 특징을 반영한 축적된 지식을 가지는 신경망 모델이 만들어질 수 있다는 점에서 그 실효성을 부분적으로 가늠할 수 있었다. (그러나, 실제 본 논문에서 제시하는 프로세스 테일러링 기법상에서 이러한 여러가지 데이터 그룹에 대한 실험이 필요한 것은 아니다.)

스텝 6. 프로젝트 상황평가

실질적인 해당 프로젝트 대상의 작업이 시작되는 단계이다. 프로세스를 테일러링하여 적용하기를 원하는 프로젝트 팀의 상황을 평가하는 작업으로 평가의 항목이 되는 것은 스텝 2 에서 정의한 프로세스 테일러링 파라미터이다. 프로세스 테일러링 파라미터의 각각의 값을 판단하기 위해 앞서 언급한 것과 같이, 프로젝트 평가 워크샵을 수행하거나, 설문을 실행할 수 있다. 프로젝트를 평가할 수 있는 참여자의 수가 매우 많을 경우, 설문을 실행하는 것이 좋으며, 그렇지 않고 한 장소에서 하루 정도 내의 토의를 거쳐 평가할 수 있는 정도의 규모일 경우엔 워크샵을 통해 집중적을 평가하는 것이 효과적이다. 토의를 통하여 각 파라미터들의 값을 하나로 결정한 경우가 아니라 설문을 통해 개개인의 평가자료를 취합해야 할 경우에는 각 개인의 평가값을 숫자로 변환하여 평균치를 파라미터의 최종값으로 결정할 수 있다. 이때, 설문참여자들의 역할에 비추어 보아 가중치를 감안하여 평균값을 산정할 수도 있다. 예를 들어 프로젝트 관리자의 평가값에는 일반 프로젝트 개발자의 평가값에 비추어 더 많은 비중을 차지하도록 가중치를 줄 수도 있다.

스텝 7. 태스크 필터링

테일러링된 프로세스를 적용하고자 하는 프로젝트 자체의 상황에 대한 평가가 이루어지면, 스텝 5 를 거쳐 미리 학습되어 있는 신경망 모델에 프로젝트 평가 결과인 프로세스 테일러링 파라미터 값들을 입력한다. 신경망 모델은 학습과정을 통해 조정된 연결강도에 따라, 주어진 특정 입력 데이터에 대해 각 태스크들의 프로세스 적응도를 0 과 1 사이의 값으로 출력한다. 예를 들어, 프로젝트의 상황이 표 3 과 같이 주어진 경우, 신경망 모델의 출력은 그림 9 와 같다. 그림 10 에서 보면, 30 개의 태스크 중 2,3,5,6,7,13,14,15,19,20,26 번 태스크의 프로세스 적응도는 99% 이상을 나타내고 있지만, 나머지 태스크들의 프로세스 적응도는 10% 미만에 머무르고 있다. 이런 경우, 10% 미만의 적응도를 보이는 태스크는 필터아웃(filter-out)되어, 해당 프로젝트에 적용되지 않는다.

본 예제에서는 별도의 역치값(threshold)을 설정하지 않아도 명확히 필터인(filter-in) 태스크와 필터아웃(filter-out) 태스크가 식별될 만큼 프로세스 적응도가 큰 차이를 보이고 있지만, 프로세스 적응도가 때에 따라서는 넓은 범위에 걸쳐 선형적으로 나타날 수도

있다. 그런 경우, 프로젝트 관리자 혹은 프로세스 엔지니어는 필터인 태스크가 가져야 할 최소 프로세스 적응도로 역치값을 설정할 수 있다. 예를 들어, 프로세스 적응도가 태스크별로 60%에서 99%에 걸쳐 골고루 출력되고 있을 경우, 프로젝트 관리자 혹은 프로세스 엔지니어가 판단했을 때 해당 프로젝트의 프로세스 정형도는 낮은 편이라고 판단되어 85%를 역치값으로 설정할 수 있다. 이런 경우, 85%를 상회하는 태스크들만이 최종적으로 프로세스 적응도 ‘ 1’ 값을 가지는 것으로 처리되어 필터인 될 수 있다. 반면, 프로세스 정형도가 높은 편으로 판단하여 65%를 역치값으로 설정할 수도 있는데, 이런 경우에는 대부분의 태스크가 필터인 되게 된다.

테일러링 파라미터	노미널(nominal)값	숫자값
프로젝트 개발형태	실험적 프로젝트	1
프로젝트 관련자 협력관계	높음	1
프로젝트규모	적음	1
지리적 분포	분산	2
개발팀의 프로세스 성숙	높음	1
개발팀의 도메인 경험	높음	1
새로운 정도	낮음	3
아키텍처 측면의 위험요소	낮음	1
기술측면의 복잡도	낮음	1
관리측면의 복잡도	낮음	1

표 3. 프로젝트 상황

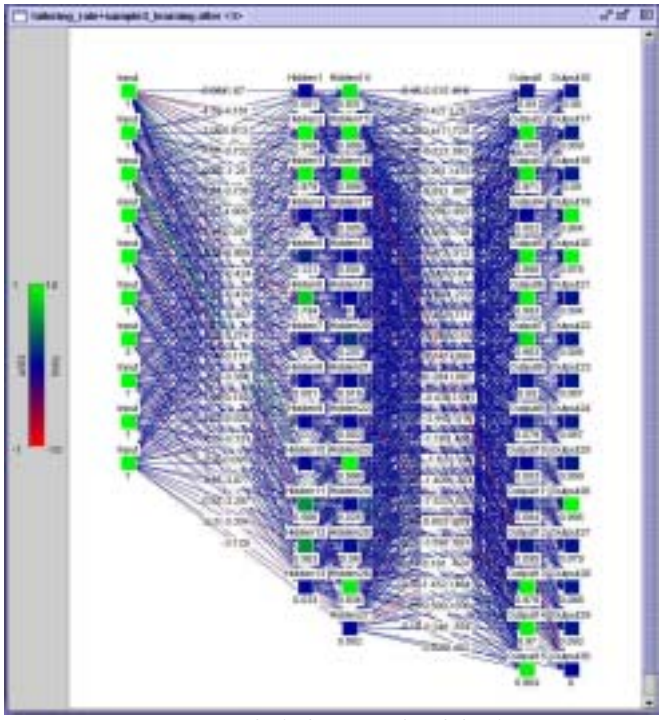

그림 10. 신경망 모델의 입출력

역치값을 설정할 경우, 최종적인 태스크 필터링에서 사람의 판단이 한번 더 들어가게 되고, 그렇지 않고 신경망 모델에서 자체적으로 프로세스 적응도를 ‘ 1’ 과 ‘ 0’ 으로 판단할 경우, 태스크 필터링은 자동으로 이루어지게 된다. 그림 11 에서 보이는 입력패턴(input pattern)은 프로세스 테일러링 파라미터의 숫자값을 나타내며, 출력패턴(output pattern)은 30 개의 태스크 각각에 대한 프로세스 적응도를 ‘ 1’ 과 ‘ 0’ 으로 양분하여 나타내고 있다. 이런 입출력 패턴쌍, 즉 테일러링 파라미터와 각 태스크들의 프로세스 적응도 쌍을 나타내는 데이터는 3.3 절에서 설명한 프로세스 피드백 결과 테일러링된 태스크의 적절성이 모두 ‘ 적절’ 한 것으로 판단된 경우에는 다시 하나의 샘플 데이터로 신경망 모델에 학습데이터로 반영되게 된다.

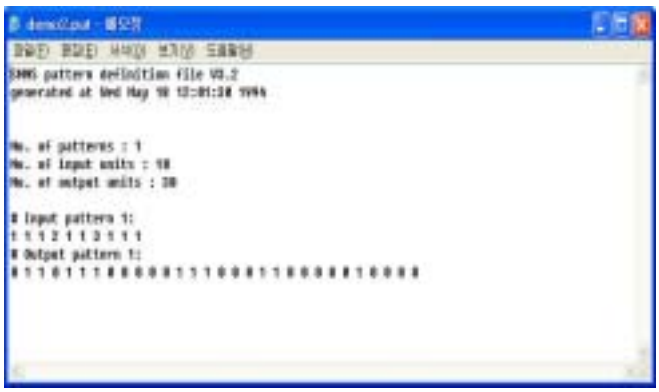

그림 11. 피드백 자료로 쓰이게 될 프로세스 테일러링 결과를 나타내는 샘플 데이터

5. 결 론

본 논문에서는 현재까지 프로세스 엔지니어들의 휴리스틱 방법에 오로지 의존하여 수행되던 소프트웨어 개발 프로세스 테일러링에 대해 신경망 모델과 같은 학습이론을 적용한 체계적인 프로세스 테일러링 기법을 제시하였다. 앞서 프로세스 필터링 단계의 실적용 예에서 볼 수 있었듯이 신경망 모델을 프로세스 테일러링에 적용함으로써 얻을 수 있는 잇점은 다음과 같이 정리될 수 있다.

- 프로세스 엔지니어에 의해 축적된 지식을 일정한 규칙 형태로 전환하고, 이러한 규칙들에 의해 생성된 샘플데이터를 이용하여 신경망 모델을 구축함으로써, 기존의 지식을 재사용할 수 있다.
- 신경망의 출력값을 결정하는 연결강도(weight)들은 프로세스 테일러링 작업의 횟수를 거듭하면서 피드백되거나 혹은 이미 보유하고 있는 테일러링 관련 과거자료(historical data)를 반영함으로써 점점 더 특정 조직의 특성을 반영할 수

있는 모델로 진화되어 간다.

기존의 몇몇 프로세스 엔지니어에 의해 이루어지던 프로세스 테일러링 작업, 특히 특정 프로세스에 적용할 프로세스와 적용하지 않을 프로세스를 구분해 내는 작업은 매우 많은 시간을 요하는 작업이었다. 그러나, 이미 학습된 신경망 모델에 해당 프로젝트가 처한 환경(environment)을 프로세스 테일러링 파라미터 의 값으로 나타내어 입력할 경우, 테일러링 대상이 되는 프로세스의 태스크에 대한 프로세스 적응도를 계산해 내기까지는 짧으면 수초, 길어도 수분 안에 완료되게 된다. 기존의 작업이 수주에 걸쳐 이루어 지던 것에 비하면 놀랄만한 비용절감의 효과가 있다고 볼 수 있다. 또한 새로운 자동화된 테일러링 방법은 실제 프로세스 엔지니어들이 내놓은 프로세스 테일러링 결과와의 비교를 통한 정확도 면에서도 평균 83% 이상을 보이고 있음을 실험을 통해 확인할 수 있었다.

또한 본 테일러링 기법은 단발적인 1 회적용에 머무는 것이 아니라 끊임없는 피드백 단계를 통해, 계속해서 테일러링 관련 지식이 쌓여나감으로써 진보되어 나갈 수 있다는 점, 신경망 모델의 특성상, 모든 테일러링 규칙을 명백히 알고 있지 못하는 상황에서도 적용해 나갈 수 있다는 점에서 일반적인 규칙기반의 전문가 시스템과도 차별성을 보이고 있다.

그러나 아직까지 본 연구는 시작단계에 있으므로 앞으로의 여러가지 연구과제들이 남겨져 있다. 몇가지 앞으로 해결해야 할 연구과제들을 살펴보면 다음과 같다. 첫째, 현재로서는 신경망 모델을 지원하는 패키지를 이용하여 시뮬레이션 해 볼 수 있는 단계로 실제 프로젝트의 프로세스 테일러링 파라미터들이 주어졌을 때 각 태스크에 대한 적용여부를 결과로 내어 놓는 부분까지는 패키지의 기능상 지원되고 있지 않다. 따라서 각각의 태스크에 대한 적용여부를 확인하기 위해서는 별도의 개발이 필요하며, 프로세스 필터링을 제외한 나머지 단계에 대해서도 실제적으로 시간상의 절약을 가져올 수 있도록 하기 위해서는 자동화 도구의 개발이 필요하다.둘째, 프로세스 테일러링 파라미터가 얼마나 태스크의 프로세스 적용유무를 판별하는 데 효과적으로 작용하는지에 대한 연구가 필요하다. 예를 들어 본 논문에서 예를 들고 있는 RUP 의 프로젝트 관리 디스플린에 속하는 태스크들은 프로세스 테일러링 파라미터중에서 " 지리적 분포" 항목과는 상관관계가 없는 것으로 보여진다. 이처럼 어떤 프로세스 테일러링 파라미터가 어떤 태스크의 프로세스 참여여부 결정에 좀 더 많은 효과를 미치는 지 등을 분석해 낼 수 있는 방안에 대한 연구가 향후 이뤄져야 할 것이다.

참고문헌

[1] C.Szyperski, Component Software : Beyond Object-Oriented programming, Addison Wesley Longman, Reading, Mass.,1998

[2] 박수용, 황만수, 박수진, 서성숙, 나호영, CMM 기반의 통합된 소프트웨어 요구사항 관리 프레임워크, 정보과학회지 제 21 권 제 4 호, pp. 33-41, 2003.4

[3] Rational Unified Process 2003

[4] CMU SEI, The Capability Maturity Model - Guidelines for Improving the Software Process, Addison Wesley, 1995

[5] Wilhelm-Schickard-Institute for Computer Science (WSI) in Tübingen, Germany, Java Neural Network Simulator (JavaNNS)

[6] ISO/IEC9126-1 Software engineering- Product quality Part1:Quality Model, ISO, 2000

[7] W. Royce, Software Project Management : A Unified Framework, Addison-Wesley, Reading, 1998

[8] J.M Benitez, J.L Castro, I. Requena, Are Artificial Neural Networks Black Boxes?, Technical Report #DECSAI-96-01-02, IEEE Trans. On Neural Networks, January 1996

[9] Khaled El Emam, Jean-Normand Drouin, Walcélio Melo, SPICE : The Theory and Practice of Software Process Improvement and Capability Determination, Addison-Wesley ,November 1997

[10] 신경망 이론과 응용(1) : 김대수, 하이테크 정보사, 1992, Page 91~142

[11] http://jongp.com/street/faqNans/ai/neural-nets/part3/section-10.html

[12] Peng Xu, Ramesh, System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on , 6-9 Pages:96 – 102, Jan. 2003,

[13] Fredrik Karlsson, Pär Ågerfalk, Anders Hjalmarsson., Method Configuration with Development Tracks and Generic Project Types, 6th CAiSE/IFIP8. International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design (EMMSAD'01), June 2001

[14] Naoufel KRAIEM, Imen BOURGUIBA, Semia SELMI, Situational Method For Information System Project., International Conference on Advances in Infrastructure for e-Business, e-Education, e-Science, and e-Medicine on the Internet, 2000.

[15] C. Rolland, N. Prakash, A. Benjamen, A Multi-Model View of Process Modeling. Requirement Engineering, Volume 4, Number 4, Springer-

Verlag, 1999
[16] Gomaa, H., Kerschberg, L. Farrukh, G.K., Domain Modeling of Software Process Models. IEEE International Conference on Engineering of Complex Computer Systems, IEEE Computer Society, September 2000