소프트웨어 요구사항 진화에서의 추적성 관리의 어려움 조사

박종열 ○, 류승희, 정세린, 이선아
항공우주 및 소프트웨어 공학전공, 경상대학교
{bjng237, tmdgml5400, rin0608 ,saleese}@gnu.ac.kr

Investigation on the difficulties of managing traceability in the evolution of software requirements

Jongyeol Park○, Seunghui Ryu, Serin Jeong, Seonah Lee
Department of Aerospace and Software Engineering, Gyeongsang National University

요 약

소프트웨어 요구사항은 소프트웨어 시스템에 대해서 사용자가 기대하는 기능, 품질 및 조건으로서, 초기에 완전하게 도출하기 어렵고, 개발 및 유지보수 중에 지속적으로 변화한다. 요구사항의 진화를 돕기 위한 주요 방법이 추적성 수립 및 관리이다. 그러나 기존의 실증적 연구는 추적성 관리가 실질적인 활용에 이르지 못한다고 보고한다. 본 연구에서는 추적성 관리를 포함한 소프트웨어 요구사항의 진화를 위한 자동화 연구를 문헌 조사한다. 또한 오픈 소스 프로젝트에서의 요구사항 변경이 발생할 때 소프트웨어 시스템에 어떤 변경이 발생하는지를 실증 조사한다. 문헌 조사와 실증 조사를 통해, 진화적 관점에서의 요구 사항 관리의 어려움의 원인을 분석하고, 도움을 줄 수 있는 자동화 연구 방향에 대해서 논의한다.

1. 서 론

소프트웨어 요구사항은 소프트웨어 시스템에 대해서 사용자가 기대하는 기능, 품질 및 조건이다. 소프트웨어 요구사항은 개발 단계 초기에 완전하게 도출하기 어렵고, 개발 및 유지보수 동안에 지속적으로 변화한다. 요구공학 프로세스는 소프트웨어 개발에서의 체계적인 요구사항의 개발을 위하여, 수집, 분석, 명세, 검증 및 관리의 단계를 제시한다. 이 중 관리의 단계에서는 주로 요구사항의 시스템 반영을 표현할 수 있는 요구사항 추적성을 수립하고 관리한다. 그러나, 지속적으로 변화하는 요구 사항을 위한 추적성 관리는 쉽지 않으며, 요구사항을 시스템에 반영하는 작업은 사람의 인지적인 능력과 결정에 대부분 의존하고 있다.

최근의 실증적 연구들은 추적성에 대한 상반된 결과를 내놓았다. 첫째는 추적성 관리가 실질적인 활용에 이르지 못한다는 보고이다. Rempel et al.은 실제 프로젝트에서 요구사항 추적성에 대한 표준 및 지침이 50%이상 적용되지 않음을 실증적으로 보였다 [1]. Cleland-Huang et al.은 추적성을 중요시하는 안전 중심의 소프트웨어 개발에서조차 바로 인증 전에 추적성을 수립하고 있음을 보고하였다 [2]. 반면 추적성의 유용성을 증명한 연구들이 있다. Asuncion et al.은 추적성 데이터를 웹을 통해 접근 가능하게 만들었을 때, 추적성 관련 작업이 2배 빨라졌음을 보고했다 [3]. Mader et al.은 실험환경에서 추적성을 사용할 때 24%의 효율성 향상과 50%의 정확도 향상을 보고했다. 이러한 연구들은 추적성의 활용이 개발자 생산성 향상에 도움을 줄 수 있으나, 실질적인 활용에 이르지 못하고 있음을 알려준다 [4].

본 논문은 상기 상반된 결과가 현 추적성 관리 방법이 소프트웨어 진화에서 발생하는 요구 사항의 변경에 대응하지 못하기 때문이라고 본다. 따라서 요구사항의 진화를 이해하고 추적성 관리를 포함한 자동화 연구의 수준을 분석하여 어떤 연구가 향후 필요한지 이해하고자 한다. 본 논문은 다음 두 가지 질문의 답을 찾기를 시도한다.

1) 요구사항의 변경에 따른 체계적인 소프트웨어 진화가 왜 어려운가?

2) 요구사항의 진화를 위한 기존 연구의 한계점과 향후 연구 기대 방향은 무엇인가?

본 논문은 이 두가지 질문에 답하기 위해서 문헌 조사와 실증 조사를 수행한다. 첫째, 문헌 조사에서는 소프트웨어 요구사항의 진화에 대한 이해 및 현 자동화 연구의 수준을 분석한다. 둘째, 실증 조사에서는 오픈 소스 프로젝트에서 요구사항 변경이 발생할 때 소프트웨어 시스템에 어떤 변경이 발생하는지를 조사한다. 이를 통해, 진화적 관점에서의 요구사항 관리의 어려움을 분석하고, 추적성 적용에 실질적으로 도움을 줄 수 있는 자동화 연구 방향에 대해서 논의한다.

본 논문의 구성은 다음과 같다. 2절에서는 소프트웨어 요구사항의 진화를 이해하기 위한 기본 개념들을 정리한다. 3절에서는 요구공학 프로세스와 자동화 연구의 문헌 조사 내용을 정리한다. 4절에서는 3절 문헌 조사를 통하여 이해한 요구사항의 체계적인 진화의 어려움을 정리한다. 4절에서는 실제 오픈 소스 프로젝트에서의 요구사항 진화를 조사한다. 6절에서는 3절부터 5절까지의 조사를 바탕으로 기존 자동화 연구의 한계점과 향후 연구 방향을 논의한다. 7절에서는 본 논문의 결론을 내린다.

2. 기본 개념

2절에서는 소프트웨어 요구 사항의 진화와 관련한 3가지 기본 개념을 정리한다.

2.1 소프트웨어 진화

소프트웨어 진화는 소프트웨어 시스템을 처음 설치한 시점부터 변경하는 모든 변경을 의미한다 [5]. Lehman은 1977년 미국의 소프트웨어 비용의 70%이상을 소프트웨어 유지보수에 사용하고 있음을 주목하고, 소프트웨어 진화의 법칙을 연구하였다[5]. 그의 연구에 따르면, 소프트웨어 시스템은 현실의 불명확한 문제를 해결하기 위해 개발되고 나면 현실의 일부가 되므로, 지속적인 변경에 노출될 수 밖에 없다. 따라서, 소프트웨어는 지속적으로 변화하며, 점증적으로 새로운 기능을 추가하면서 복잡도가 높아진다[5].

2.2 요구 공학 프로세스

요구 공학 프로세스는 관련자들로부터 소프트웨어에 대한 요구 사항을 수집하여 기대 수준에 부합하는 소프트웨어 시스템을 만들기 위한 요구 사항 관련 프로세스이다[6, 7]. 요구 공학 프로세스는 수집, 분석, 명세, 검증, 관리의 5가지 단계로 구성되며, 각 단계의 요약은 다음과 같다 [7]:

- 요구사항 수집: 고객, 기술, 환경에서의 요구사항을 수집한다.
- 요구사항 분석: 개발 단계별로 발전하는 요구사항 할당과 비즈니스, 사용자, 시스템, 소프트웨어 수준별로의 상세화를 진행한다.
- 요구사항 명세: 요구사항마다의 고유한 아이디 명시하고 사용사례와 품질 속성을 포함한 요구사항을 기술한다.
- 요구사항 검증: 시스템 수준의 요구 사항과 상세 수준의 요구 사항의 일치 등을 검토한다.
- 요구사항 관리: 추적성 수립을 통하여 소프트웨어 요구 사항의 생명 주기를 관리한다.

2.3 소프트웨어 추적성

소프트웨어 추적성은 “고유하게 식별 가능한 소프트웨어 공학 산출물을 다른 것과 연관시키는 기능”이다 [8]. 추적성을 수립하기 위하여 주로 요구사항 추적성 매트릭스(Requirement Traceability Matrix 혹은 추적성 테이블)를 사용하는데, 추적성 매트릭스란 개별 기능 요구사항과 다른 시스템 산출물 간의 논리적 맵핑을 설명하는 표로서, 시스템 요구사항, 소프트웨어 요구사항, 설계 항목, 코드 항목, 시험 항목으로의 맵핑을 순차적인 배열로 표현한다 [8]. 전형적인 추적성 매트릭스는 표 1과 같다.

[표 1] 전형적인 추적성 매트릭스 ([8,9])

요구사항	설계모듈	구현파일	테스트항목
RE001	DE003	IM002	TE002
…	…	…	…

이러한 추적성 맵핑 관계는 소프트웨어 시스템의 크기와 복잡성에 따라 기하 급수적으로 커지는 문제점이 있다.

3. 문헌 조사

3절에서는 소프트웨어 요구사항의 진화 및 자동화 연구를 문헌 조사한다. 표 2는 조사한 결과를 보여준다. 표 2에서는 연구를 개발과 진화로 나누어 요구사항 개발과 관련된 문헌은 일반 폰트로, 요구사항 진화와 관련된 문헌은 볼드체에 밑줄로 표시하였다. 세로축의 상세 분류는 기법 및 도구, 문헌 조사, 실현상 조사로 나눈다. 가로축으로는 먼저 요구사항 명세 전과 명세 후 활동으로 구분하고, 상세 분류는 수집, 분석, 검증, 관리의 활동으로 구분한다. 해당 표를 바탕으로 3.1와 3.2절에서는 기법 및 도구, 3.2절에서는 문헌 조사, 3.3절에서는 실현상 조사로 나누어 요약, 논의한다.

3.1 요구사항 수집, 분석, 검증의 자동화 기법 및 도구

요구사항 수집, 분석, 검증에 있어서 소프트웨어 진화와 관련된 연구는 요구사항 수집에서 있었다. 요구사항 분석 및 검증에서 소프트웨어 진화와 연동한 연구는 우리가 찾는 문헌 내에서는 없었다.

3.1.1 요구사항 수집의 자동화 연구

2013년도 논문 [10]은 소프트웨어 진화에 반영해야 하는 요구 사항의 자동 수집에 초점을 두고 있다. 이를 위해 사용자 코멘트를 분석해서 요구사항을 자동으로 도출하기를 제안하였다. 그리고 부정적인 내용에 대한 주제를 도출하기 위해 Aspect and sentiment unification Model(ASUM) 데이터 마이닝 기법을 사용하였다. 논문 [10]은 Manual 기법보다는 상대적으로 시간을 단축할 수 있다는 부분을 확인하였다. 단지, 일반적인 분석기법, K-median Clustering 기법과 비교하여 ASUM 적용 성능은 비슷하였다. 논문 [10]은 사용자 코멘트를 분석하는 토픽 모델링기법의 유용성을 처음으로 평가한 논문으로, 사용자들이 실제 요구하는 사항들을 분석함으로써 소프트웨어 진화에 반영되어야 하는 요구사항을 편리하게 도출할 수 있도록 돕는다.

3.1.2 요구사항 분석의 자동화 연구

2016년도 국내 논문 [11]은 자동화된 요구사항 우선 순위 분석을 제안하였다. 논문 [11]은 요구사항의 수가 많은 경우, 적용하기가 힘든 확장성 문제와 이해관계자의 가치판단에 노출되는 편향성 문제를 지적하였다. 그리고 정제된 요구사항과 정제되지 않은 요구사항 사이의 연관성을 계산하여 정제된 요구사항의 우선순위를 정하는 ToMSN 기법을 제시하였다. 또한 출입 제어 시스템 프로젝트의 정제되지 않은 요구사항과 정제된 요구사항에 대하여 ToMSN 기법을 적용한 사례를 제공하였다. 논문 [11]의 신규성은 중립적이고 풍부한 요구정보 데이터를 적극적으로 활용하는 기법을 선보인데 있다. 그러나, 요구사항의 진화에서 나타나는 우선순위 갱신 문제를 다루지는 않는다.

[표 2] 문헌 조사표

<table>
<tr><td rowspan="4">개발 및 진화</td><td>실현상 조사</td><td></td><td>Robinson et al. 2015
Schneider et al. 2016</td><td></td><td>Gotel et al. 1994
Asuncion et al. 2007
Rempel et al. 2014
Mader et al. 2015</td></tr>
<tr><td>문헌조사</td><td colspan="3">Ernst et al. 2014
Achimugu et al. 2014</td><td>Cleland-Huang et al. 2014</td></tr>
<tr><td>기법 및 도구</td><td>Carreno et al. 2013</td><td>Jang et al. 2016</td><td>Ko et al. 2016</td><td>Kim et al. 2010
Kim et al. 2012
Eom et al. 2015
Cleland-Huang et al. 2003
Klock et al. 2011
Cleland-Huang et al. 2014
Saito et al. 2016
Garcia et al. 2016</td></tr>
<tr><td colspan="2"></td><td>수집</td><td>분석</td><td>검증</td><td>관리</td></tr>
<tr><td colspan="2"></td><td colspan="2">명세 전</td><td colspan="2">명세 후</td></tr>
</table>

3.1.3 요구사항 검증의 자동화 연구

2016년도 국내 논문 [12]는 요구사항 패턴에 기반한 누락 요구 사항의 검증을 제안하였다. 논문 [12]에서는 요구사항 수집에 있어서 항목 누락이 될 수 있는 문제에 대하여, 요구사항 시나리오에 대해 기계학습을 적용하여 소프트웨어 요구사항 패턴을 추출하는 기법을 제시하였다. 평가를 위해서는 여덟 개의 프로젝트의 요구사항 시나리오에서 패턴을 시범적으로 도출하여 요구사항 패턴 추출의 자동화가 가능함을 보였다. 논문 [12]의 결과는 요구사항을 수집할 때 요구사항의 누락을 방지할 수 있도록 가이드라인을 생성하는데 활용할 수 있다. 논문 [12]는 요구사항 수집에 초점으로 두는 반면, 그 요구사항의 진화에서 발생하는 일관성 검증 문제를 다루지는 않는다.

3.2 요구사항 관리의 자동화 기법 및 도구

요구사항 관리는 2.3절에서 언급한 것과 같이 추적성 수립을 통한 방법을 주로 사용한다. 그러나 추적성 수립 이후의 소프트웨어 진화 방법에 대응하는 부분에 대해서는 자동화되어 있다고 보기 힘들다.

3.2.1 요구사항 관리의 기법 연구

2010년도 국내 논문 [9]는 요구사항 변경 시에 추적성 테이블의 요구사항 ID를 어떻게 관리할지를 제안하였다. 논문 [9]는 기존 추적테이블 연구는 변경관리 방법 및 추적의 효과를 구체적으로 제시하고 있지 않고 있음을 지적하였다. 또한 변경 영향 추정연구는 추정방법이 복잡하여 실용성에 한계가 있음을 주목하였다. 논문 [9]는 변경 요구사항에 대해서 새로운 ID를 부여하자고 제안하였다. ID의 변경요구사항에 대해서 변경의 원천 요구사항을 알 수 있도록 기재함으로써 변경율 및 변경영향도를 기존 연구에 비해 용이하게 추정할 수 있다고 주장하였다. 그러나, 이러한 제시 방법은 기존 추적성 방법에 변경 요구 사항을 어떻게 추가할 것인가의 개선 제안이라는 한계점을 가지고 있다. 즉, 요구사항 진화에서 발생하는 다양한 영향 요소 및 요구사항 변경에 영향을 받는 소프트웨어 산출물의 다양한 내부 관계(예: 요구사항 간의 계층 등)를 고려하지 않으며, 이에 따라 요구사항 진화를 통합 관리할 수 있는 방법이라고 보기 어렵다.

2012년도 국내 논문[13]은 기존의 추적 기법은 추적의 정도에 대한 정량화된 평가 방법과 판정 기준이 없다고 지적하였다. 따라서 정략적인 추적의 정도를 수치로 표현하기 위해서 요구사항 추적 모델에서의 선행항목이 후행항목과 관련되는 정도를 나타내는 요구사항 전개도 정도를 수식으로 일반화 하여 추적성을 정량적으로 나타내었다. 또한 산림청 정보화 사업 16개를 대상으로 사업 사례분석을 통해 요구사항 반영의 적정성 판정 사례를 보였다. 논문[13]은 저자들의 과거 연구보다 좀 더 다양한 추적 환경을 고려하여, 추적성의 정도를 정량화하였다. 그러나 소프트웨어 요구사항의 진화와 연관시키지는 않았다.

2014년도 국내 논문 [14]는 제품라인에서의 요구사항과 아키텍처 사이의 추적성 구축에 대한 연구를 진행하였다. 논문 [14]에서 주목한 문제로 소프트웨어 요구사항과 아키텍처 설계 사이의 추적성 구축은 요구사항과 아키텍처 구성 요소 사이의 다대다 관계와 아키텍처 계측으로 고려해야 할 요소가 많으며, 제품라인으로 가면 복잡도가 더 증가한다. 논문 [14]에 서는 이를 해결하기 위해 제품라인 및 아키텍처 계층을 고려한 추적성 수립을 위한 재귀적인 방법을 제시하였다. 사례 연구로는 전자레인지 제품라인 개발의 추적성 모델링을 진행하였다. 논문 [14]는 소프트웨어 제품 라인의 진화를 돕기 위하여 요구 사항과 소프트웨어 아키텍처를

맵핑하는 추적성 수립 방안을 제시하였다. 그러나, 추적성 수립일 경우로 국한되며, 이후의 요구사항의 진화에 대응하는 연구는 아니다.

3.2.2 요구사항 관리의 도구 연구

소프트웨어 추적성 수립을 지원하기 위한 다양한 요구사항 관리도구 들이 있다 [15]. 그 중 가장 유명한 상용화 도구는 DOORS이다 [16]. DOORS의 주요 기능은 요구사항 간의 혹은 요구사항과 다른 산출물 간의 추적성(Traceability) 명시와 히스토리, 베이스라인, 교환 등이다. DOORS의 문제점은 프로세스를 수정한 후 이를 반영할 때 자유롭게 변경 할 수 없는 제약이 있다. 많은 도구가 개발되었음에도 실질적인 요구사항 추적성의 적용과 활용이 성공적이지 않음은 오랫동안 인식되어 왔다 [1, 2, 17]. 이를 해결하기 위하여 진화적 측면의 요구사항에 대한 변경 전파를 위한 연구 및 제안들이 있었다 [8, 18, 19, 20, 21].

2003년도 논문 [18]은 요구사항의 변경 전파를 위하여, 이벤트 기반의 추적성 전파 시스템을 제안하였다. 제안 시스템에서는 요구사항의 변경을 변경 이벤트로 간주하고 이에 영향을 받는 산출물을 이벤트를 받아야 하는 구독자로 간주하였다. 그리고 요구사항의 변경이 발생할 때 이벤트 메시지에 수정해야 할 사항을 담아 넘기면 영향을 받는 산출물에 대해 해당 사항을 반영하였다. 사례 연구로 M-Net이라 는 웹 기반 학회 시스템을 대상으로 제안 시스템을 적용하여 활용 가능성을 보여 주었다.

2011년도 논문 [19]는 수동으로 산출물간에 추적성 수립을 하는 것은 최신 상태의 유지가 힘들다는 부분을 지적하였다. 해결 도구로 제안한 Traceclips는 자연어 분석과 정보 검색 기술을 활용하여 추적성 링크를 유추하여 사용자에게 보여주고 사용자 친화적인 인터페이스를 제공하여 개발자가 추적 가능성 링크를 관리 가능하게 한였다. 또한, 평가를 위해서 루신 프로젝트를 대상으로 75개의 추적성 링크를 자동, 유추하여 정확도가 약 40%, 재현도가 약 30%임을 보여주었다. 논문 [19]는 요구사항의 추적성의 갱신에 관하여 추적성을 자동으로 수립하는데 공헌하였다.

2014년도 논문 [20]은 추적성 데이터가 종종 불완전하고 부정확하며 중복되고 상충되고 더 이상 유효하지 않아 신뢰성이 낮음을 지적하였다. 이를 위해 추적성 도구에서 링크를 신뢰할 수 있는 추적 링크와 신뢰할 수 없는 링크로 구분하기를 제안하였다. 또한, 신뢰할 수 있는 링크들에서 더 이상 신뢰할 수 없는 링크를 끊임없는 제거하기를 제안하였다. 마지막으로, 추적 링크를 체계적으로 구축하거나 재구성하는 데 사용되는 신뢰할 수 없는 추적 링크 증거를 모으기를 제안하였다. 논의를 위해 두 개의 변경 시나리오를 제시하였으나 실제 사례를 보여주지는 못했다. 단지, 사용자가 추적 링크를 변경할 때, 링크들을 점검하는 활동을 언제할지 선택할 수 있다는 이점을 논의했다.

2016년도 논문 [21]는 요구사항 진화를 위한 시각화 도구를 제시하였다. 제시 도구는 정의된 일곱가지의 요구사항 진화 이벤트를 사용하여 요구사항 진화 차트를 시각화하였다. 도구 유용성을 평가하기 위해서는 문서 관리 및 승인 시스템의 개발자를 두 개의 그룹으로 나누어 활용하도록 하였다. 그 결과, 도구를 활용한 그룹이 도구를 활용하지 않은 그룹보다 변경된 요구 사항에 대해서 22프로 더 정확하고 15프로 더 신속하게 이해함을 확인하였다. 이 때의 추적성은 요구사항 간의 진화에 초점을 맞추고 있으며 시스템 반영 추적성은 시각화하지 않는다.

2016년도 논문 [8]은 요구사항의 추적성을 소프트 웨어 구현물에 직접 매핑하는 도구를 제시하였다. 논문 [8]에서는 산업계에서 기존 요구사항 관리 도구를 50% 밖에 사용하지 않음을 주목하였다. 그 이유로 기존의 추적성 도구가 산업계의 요구에 부적절하다고 보았다. 이에 따라 웹 애플리케이션의 요구사항과 구현 산출물간의 매핑할 수 있는 도구를 제안하였다. 제안 도구에서는 웹 페이지와 HTML 요소 및 컨트롤과 같은 구현 산출물을 클릭한 후, 기능 요구 항목을 드롭박스 에서 클릭하면, 해당 매핑 관계를 나타내는 추적성 링크를 데이터베이스에 저장한 후, 사용자가 원할 때 해당 추적성 매트릭스를 시각화하여 보여준다. 제안 도구은 추적성 링크를 생성하기 위해서는 인간 상호 작용을 지원하며, 요구사항과 실질적인 산출물인 웹 요소 사이의 직접적인 매핑을 하여 그 결과를 시각화하는 특성이 있다. 그러나 복잡한 소프트웨어 분석, 설계, 구현 항목들의 맵핑 관계를 간과한 측면이 있다. 또한 사람이 넣은 것 이상의 매핑 관계를 유추하여 보여주는 부분이 없다.

3.3 문헌 조사 연구

요구공학 분야에서의 문헌 조사 연구는 비교적 최근에 진행된 것으로 보인다. 문헌 조사는 주로 과거의 연구들을 정리함으로써 연구 방향을 수립하기 위해 진행되었으며, 우리가 찾은 3개의 문헌 조사는 요구분석 [22], 요구사항 진화 [23], 추적성 연구 [2] 각각에 초점을 맞추고 있다. 그러나 본 논문에서 추구하는 진화적 관점에서의 요구 사항 관리의 어려움에 대한 논의를 제공하지는 않는다.

2014년도 논문 [22]는 요구사항 분석의 자동화(3.1.2절 참조)와 관련된 요구 사항의 우선순위 결정 기법들을 모아 분석하였다. 분석 방법으로는 체계적인 문헌 검토 기법을 이용하여 기존의 우선순위 결정 기술에 대한 논문을 조사하고 분석하였다. 그 결과 분석 계층화 과정(Analytic hierarchy process), 과학적 연구 개발 방법(Quality functional deployment), 게임 계획(Planning game)등이 자주 연구, 활용됨을 밝혔다. 또한 기존 우선순위 결정 기술들은 확장성의 부족, 요구사항 진화 중 순위 업데이트 처리 방법, 이해 관계자 사이의 조정, 요구사항 종속성 등의 한계가 있음도 보였다. 논문 [22]에서 요구사항의 진화와 관련하여서는 우선순위 갱신이 어려움을 논의한 파트가 있었다. 단, 요구 분석의 우선 순위 기법에 대해 초점을 두며, 요구 진화에 따른 관리의 어려움을 이해하고자 하는 우리의 시도와는 초점이 다르다.

2014년도 논문 [23]은 요구사항 진화에 대한 문헌 조사를 진행하였다. 논문 [23]은 요구공학이 요구사항의 변경을

포함해야 함을 지적하였다. 이를 위해 요구사항의 변경이 소프트웨어 아키텍처와 구현에 미치는 영향을 분석하기 위한 프로세스 모델을 찾아내고자 하였다. 요구 사항의 변경을 이해하기 위해 문제를 요구사항 R, 요구사항 명세서 S, 그리고 도메인 지식 W의 관계로 정형화 하였다. 다음, 정형화한 문제를 바탕으로 각 기존 연구의 형태를 모델링하여 요구사항의 진화에 대한 프로세스 프레임워크를 제시하였다. 해당 논문은 요구 사항이 변경될 때의 처리 프로세스를 모델링하였으나, 요구사항 변경의 어려움 혹은 복잡성의 원인에 대한 분석은 하지 않았다.

2014년도 논문 [2]는 추적성과 관련한 과거 10년간의 연구를 분석함으로써 연구 방향을 찾고자 하였다. 논문 [2]도 추적성이 중요하다고 인식되고 있으나, 실제적인 적용이 힘든 점에 주목한다. 특히 안전 중심의 소프트웨어 개발에서조차도 추적성 수립이 인증 바로 전에 만들어지고 있어 실질적인 활용이 되지 않고 있는 문제를 지적하였다. 문헌 조사를 위해서는 세가지 관점, 목표, 프로세스, 내부구조를 정의하고, 이 세가지 관점에 기반하여 과거 10년간의 추적성 연구들을 분석하였다. 그 결과, 일반적인 추적성 연구가 50%가 넘고, 특정한 목표를 가정한 추적성 연구는 약 25%가 됨을 확인하였다. 또한 추적성 생성 연구에 40% 이상의 비중이 몰려 있음을 확인하였으며, 마지막으로 추적 알고리즘에 50%의 연구의 노력들이 몰려 있음을 확인하였다. 마지막으로 문헌 조사를 바탕으로 향후 연구 주제를 논의하였다. 논문 [2]는 도입부에서 지적한 추적성의 실제적인 적용이 왜 힘든지에 대한 논의는 하지 않았다.

3.4 실현상 조사 연구

요구공항 분야에서의 실현상 조사 연구는 주로 요구사항 진화의 현상에 초점을 맞추고 있다. 추가로 기존에 개발된 추적성 지침 및 도구의 실제적인 적용 및 효과에 대한 검증 연구가 있었다.

3.4.1 요구사항 진화의 실현상 연구

2015년도 논문[24]는 오픈 소스 프로젝트에서의 요구공학의 적용을 분석했다. 논문 [24]에서는 기 제안된 six-V 측정 모델을 바탕으로 오픈 소스 프로젝트를 수치적으로 분석하였다. 여기서 six-V는 Volume(양), Veracity(진실성), Volatility(휘발성), Vagueness(모호성), Variance(분산), Velocity(속도)를 의미한다. 논문 [24]는 sourceForge에 있는 31가지의 프로젝트에 대해서 각 V의 속성이 있는지 가설을 세워서 각 V의 요구사항 속성을 수치적으로 표현하고 통계적으로 가설을 만족하는지 검증하였다. 논문 [24]는 지속적으로 변화하는 요구사항에 대하여 수치적으로 분석하여 객관적인 자료를 도출시킬 수 있음을 보여주었다. 하지만, 소프트웨어 요구사항에서의 진화가 왜 어려운지, 어떻게 그 어려움을 극복할 수 있는지에 대한 논의를 제공하지는 않는다.

2016년도 논문 [25]는 요구사항의 진화의 매커니즘을 제시하기 위해, 한 회사의 소프트웨어 시스템의 생명주기 전반에 걸친 요구 사항의 진화를 분석하였다. 이를 위해 인터뷰 (2013 년 및 2014 년), 문서 수집 (internal slide decks, 스프레드 시트, 공개 문서), 비공식 임시 질문 (전화, 전자 메일, 비공식 현장 미팅 및 즉석 질문) 등을 수행하였다. 그 결과, 문맥적인 요소를 바탕으로 이벤트가 발생하면, 이러한 이벤트들이 요구 사항과 소프트웨어 시스템의 진화를 발생시키고, 이러한 진화된 결과물을 바탕으로 경험이 축적되어 또 다른 이벤트를 발생시키는 메커니즘을 밝혀 내었다. 또한, 구성을 설정하거나 유지 보수성을 강화시키는 요구사항들이 소프트웨어 생명 주기가 흐를 수록 좀 더 중요한 우선순위를 차지함도 밝혀 내었다. 그러나, 논문 [25]는 한 회사의 전사적 시스템 개발 및 사용의 경험 데이터를 관찰한 것으로, 겉으로 보이는 요구 사항에 대한 결과를 도출했을 뿐, 요구사항 진화의 어려움에 대한 요인 분석은 제공해 주지 않는다.

3.4.2 추적성 지침 및 도구 적용 연구

요구사항 추적성이 요구사항의 진화를 지원하지 못하는 문제점에 대해서는 오래 전부터 인식되어 왔다. 산업계에서 추적성을 활용하기에는 부적절한 이유를 파악하기 위해서 1994년 이미 실질적인 사용 조사를 진행한 바 있다. 논문 [17]은 그 당시 이미 100가지가 넘는 요구사항 추적성 도구의 강약점을 파악하고 100여명을 대상으로 설문조사와 인터뷰를 진행하여 추적성 활용의 문제점을 조사하였다. 논문 [17]의 결론은 첫째, 사전 RS 추적성에 대한 연구를 수행할 필요가 있으며, 둘째, 의사소통을 원활하게 하기 위해, 구체화한 그리고 정제한 요구 사항을 신속하게 찾아 접근할 수 있는 지속적인 능력이 필요하다는 것이었다. 25년이 지난 현 시점에서는 3.1.1절과 같이 사전 RS 추적성에 대한 자동화도 제시되고 있어, 해당 논문에서 주장한 문제점을 위한 솔루션이 나오고 있으나, 아직 상기 지속적인 능력에 대해서는 연구자들이 합의한 해결 방법을 찾지 못하고 있다.

2007년도 논문 [3]에서는 Doors와 같은 전문화 된 추적 도구가 채택 된 회사에서도 툴의 완성도, 좁은 초점 및 다른 도구와의 상호 운용성 부족으로 인해 추적성 문제가 존재함을 보고하였다. 따라서 이를 경제, 기술, 사회적 관점에서 보완한 실제 회사 제약 조건 내에서 최종 추적성(end-to-end tacebility) 소프트웨어 추적 도구를 개발하여 제공하였던 사례를 보고하였다. 개발한 도구는 웹을 통해 원격 사용자를 포함 조직의 모든 구성원이 즉시 엑세스 가능하게 하였다. 도구 사용 사례를 위해서 산업 소프트웨어를 전문적으로 개발하는 IT회사(직원 250명)의 실제 프로젝트 데이터로 저장소를 채워 개발한 도구를 사용하여 테스트를 진행하였다. 그 결과 추적성 관련 작업이 2배 빨라졌음을 보고하였다. 이는 추적성 데이타를 활용할 때 생산성이 높아질 수 있음을 보여주는 데이터이지만, 과거 데이터로 테스트를 진행한 점에 있어서 평가가 제약되는 점이 있다.

2014년 비교적 최근 논문 [1]에서도 여전히 실제 프로젝트 상황에서 요구사항 추적성에 대한 지침이 적용이 되지 않고 있음을 실증적으로 보여준다. 논문 [1]에서는 이러한 평가를

위하여, 요구사항 추적성 가이드라인으로부터 정형 모델을 만든 다음 프로젝트 데이터로부터 해당 정형 모델로의 맵핑을 분석한다. 이후 정형화된 추적성 문제 규칙을 적용하여 추적성 문제와 타입을 명시한다. 논문 [1]은 해당 평가 모델로 5가지 기술지침에 대해 7가지 안전성 소프트웨어 시스템의 추적성을 평가했으며, 대부분의 프로젝트가 50퍼센트 이상 추적성 가이드라인에 부합하지 않음을 보고한다. 논문 [1]에서는 가이드라인에 의해 규정된 것과 실제로 구현되는 것 사이에 차이가 있음을 객관적인 데이터로 보여준다.

2015년 논문 [4]에서는 요구사항 추적성의 유용성에 대한 평가가 발표된 적이 없음을 지적하였다. 이를 객관적으로 증명하기 위해 사용자 실험을 진행하였다. 실험 방법으로 2개의 제 3자 개발 프로젝트에서 실무자부터 다양한 경험이 많은 학생들까지 총 71명을 대상으로 추적성을 적용한 그룹과 적용하지 않은 그룹으로 나누어 실제 유지 보수 작업을 다시 수행하도록 하였다. 평가를 위해서는 2x2x4x4 계승 설계를 사용하였고, 두 프로젝트의 각각에 대해 4 개의 작업이 정의하고 각각 추적성을 사용하거나 사용하지 않고 작업을 수행하였다. 그 결과 추적성을 사용한 그룹이 사용하지 않은 그룹에 비해 평균적으로 24% 빠르게 작업을 하였고, 50% 더 정확한 해결책을 만들었음을 통계적으로 보여주었다. 이러한 연구는 추적성이 주는 생산성 향상을 객관적으로 보여주는 자료로 유용하다. 문제는 실험 연구이기 때문에 실질적인 추적성 수립 및 진화에서 발생하는 이슈는 배제한 실험이라는 것이다.

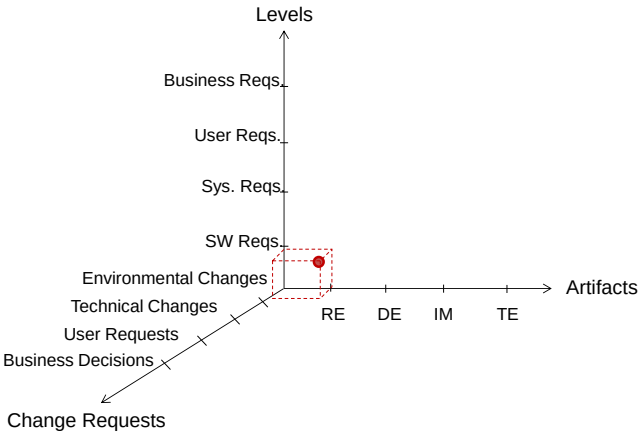
4. 요구사항의 진화에서의 어려움 요인 논의

4절은 문헌 조사를 통해 파악한 다음 질문의 논의를 정리하고자 한다: “요구사항의 변경에 따른 체계적인 소프트웨어 진화가 왜 어려운가?”

우리는 요구 사항의 변경과 연관된 다양한 측면이 존재하며, 요구 사항의 변경에서 발생하는 활동은 일반 코드의 변경과 유사하고, 마지막으로 소프트웨어 개발 산출물의 어느 항목에서 변경이 발생하든지 이를 반영할 수 있어야 한다는 사실을 발견하였다. 각각의 논의를 세부 절로 정리한다.

4.1 요구사항의 다차원적인 진화

논문 [6]의 요구공학 분석에서는 요구사항과 관련된 여러 다른 측면을 기술한다. 여기에서 도출할 수 있는 사실은 요구사항의 변경을 다루기 위한 여러 측면이 존재한다는 것이다. 그림 1은 그 중에 세 가지 측면을 차원으로서 표현해 본 것이다. 그림 1에서 변경요청의 측면에는 사용자의 요청, 기술적인 변화, 환경적인 변화 등이 있다. 요구사항 수준의 측면에서는 상위 수준의 요구 사항으로부터 하위 수준의 요구 사항이 존재한다. 예를 들어 사용자 수준의 요구 사항과 소프트웨어 시스템의 요구 사항으로 세분화될 수 있다. 마지막으로 변경영향 항목의 측면에서는 요구 항목의 변화로 영향을 받는 산출물로 개발, 구현, 시험까지의 각각의 항목이 될 수 있다 (문헌 [6]의 II.A.2절 참조).



[그림 1] 요구사항에서의 진화와 연관된 항목의 차원

논문 [14]에서도 비즈니스 결정사항을 포함한 문맥적 요소로 표현된 요구사항의 변경 요청을 어떻게 요구 사항 및 시스템에 반영하는지에 대한 메커니즘을 논의하고 있으나, 세분화된 요구사항과 산출물의 반영은 고려하고 있지 않다. 논문 [26]에서는 “multi-faceted” 추적성 문제를 언급하며, 추적성의 어려움에 많은 요인들이 연루되어 있음을 지적한다. 이를 통해 요구사항의 다차원적인 진화를 다루는 것이 어렵다고 본다.

4.2 요구사항의 변경 타입

4.1절의 다차원에서 발생하는 요구 사항의 진화에는 시간의 흐름이라는 이라는 추가 측면이 존재한다. 예를 들어 그림 1에서 환경적 변화 요청에 따라 요구명세서의 소프트웨어 요구 사항이 변경되었다면, 해당 변경이 무엇인지를 명시하는 것이 요구 사항의 진화를 이해하는데 도움을 줄 수 있다.

논문 [18, 21]은 요구사항의 변경 타입을 다음의 7가지 변경 타입으로 제시한다.

- 생성(Create): 새로운 요구사항을 생성한다.
- 비활성화(Inactivate): 요구사항을 비활성화한다.
- 수정(Modify): 요구사항의 속성이 일부 바뀐다.
- 세분화(Refine): 하나의 요구 항목이 항목의 의미가 변하지 않고 추가적인 요구사항으로 상세해진다.
- 분할(Decompose): 하나의 요구 사항이 여러 요구 사항으로 분할된다.
- 통합(Merge): 여러 요구 사항이 하나의 요구 사항으로 통합된다.
- 대체(Replacement): 하나의 요구 항목이 폐지되고 다른 요구 사항으로 바뀌는 것을 뜻한다.

상기 변경 타입을 검토하면 요구사항의 변경 또한 일반적인 변경과 같이 추가, 삭제, 수정이라는 부류가 있으며, 해당 부류들을 7가지로 세분화할 수 있음을 볼 수 있다. 이를 통해 요구사항의 진화에 있어 다차원에서 발생하는 변경에 대해서 변경 타입을 고려하여 소프트웨어 산출물에 반영하는 패턴을 볼 수 있다고 유추한다.

4.3 변경의 순방향, 역방향 전파

4.1절의 다차원적인 측면에서 변경은 여러 측면의 조합 중 어디에서나 발생할 수 있음을 파악하였다. 또한 4.2절의 변경 타입을 통해 요구사항의 변경이 일반적인 변경의 타입을 따름을 파악할 수 있었다. 이는 우리가 요구사항의 변경에 한정하지 않고, 어느 항목에서 변경이 일어나든 전파가 가능해야 한다는 것을 뜻한다.

반면 표 1의 추적성 매트릭스는 순방향 전파를 가정하고 있다고 볼 수 있다. 즉, 요구, 설계, 구현, 테스트 항목에 있어, 선행항목의 변경이 발생하면 어떤 후행항목이 영향을 미치는지를 분석할 수 있는 정보의 명시를 주 대상으로 한다. 예를 들어, 요구명세서의 요구 항목을 변경하면, 이를 구조 설계서의 모듈에 반영하고, 다음으로 상세 설계서 및 코드에 반영을 하는 방식이다. 우리는 이러한 가정이 틀렸다고 본다. 변경은 모든 소프트웨어 산출물 항목 어디서든 발생할 수 있다고 보는 것이 더 실제적이다. 따라서 순차적인 할당이 아니라 어디서든 발생하는 항목의 변경에 따른 전파를 지원하는 추적성 수립 형태가 있어야 한다고 본다. 따라서 선행 항목의 변경이 후행 항목에 미치는 영향을 분석하고 또한 후행 항목의 변경이 선행 항목에 미치는 영향을 분석할 수 있는 순방향, 역방향 전파의 영향 분석이 가능해야 한다.

5. 실 사례에서의 요구 사항의 진화 조사

4절에서는 실 사례에서의 요구 사항의 변경에 대하여 조사한다. 조사 대상은 오픈 소스로 운영되는 코드 편집기로써 GitHub가 개발한 ATOM 에디터이다[1]. ATOM 에디터는 2015년 6월 25일에 정식 버전이 출시되었으며 커뮤니티와 Git hub 사이트를 통해 사용자들과 소통이 활발하게 이루어지고 있다. 오픈 소스의 특성상 이슈 게시판에 사용자이 요구사항을 요청하기도 하고 토론도 진행되며, 개발자들은 요청 내용을 적극적으로 수용하여 패치를 배포한다.

본 논문의 저자 4명은 GitHub 싸이트의 ATOM 이슈 게시판에 올려진 사용자들의 요청사항을 조사하였다. 조사 방법은 각자 ATOM에디터에서 기능 하나를 임의로 선택하였고, 이슈 게시판의 검색과 필터링 기능을 사용하여 시스템 반영이 완료가 된 요청사항만을 조사 대상으로 하였다. 각 요청사항에 대한 이슈 보고서를 읽으면서 개발자 사이에서 어떠한 논의가 오갔는지와 결과적으로 어떠한 코드 변경이 발생했는지 조사하였다. 조사 결과를 요구사항 타입(Requitemnet Type) 대비 시스템반영 타입(Code Change Type)으로 나누어서 분류하였다

요구사항의 변경타입은 앞선 4.2절에서 언급한 7가지를 참조하였다. 시스템반영 타입(Code Change Type)은 직접 프로그램의 코드가 어떻게 변경되었는지를 말하며, Add, Delete, Invalidate, Reject, Change(Modify, Replace, Deprecate, Move, Refine, Merge) 7종류로 분류하였다. 각 타입의 의미는 다음과 같다. 추가(Add)와 삭제(Delete)는 변경된 요구에 따라 새로운 코드를 추가하거나 삭제한다. 무효화(Invalidate)는 요구를 반영하기로 하였지만 그럴 필요가 없어진 사항들이다. 거절(Reject)은 요구를 거부한다. 변경(Change)은 수정, 변경, 이동, 합병 등으로 세분화한다.

표 3은 우리가 조사한 결과이다. Random하게 조사한 결과 요구 사항의 변경에 있어 생성(Create), 수정(Modify), 세분화(Refine), 대체(Replace)의 결과를 찾아낼 수 있었다. 서로 다른 타입의 요구사항의 예를 제시하면 다음과 같다.

- 생성(Create): Issue#8939에서는 Top과 Bottom bar를 에디터에 추가해줄 것을 사용자가 제안하였다. 이에 대한 코드 반영은 추가(Add)와 수정(Modify)로 발생했다. 개발자들은 기본 에디터의 틀을 변경한 뒤, Top과 Bottom bar를 나타내는 요소를 추가해주었다. 요구사항 생성에 있어서는 대부분의 코드 반영에 추가(Add)가 존재하였다.
- 수정(Modify): Issue #11486에서는 CTRL+W로 작동하는 화면 닫기 기능을 CTRL+F4로 변경해 달라는 요구가 있었다. 코드 반영은 추가(Add)로 되었는데, 이유는 CTRL+W로 변경을 원하는 사용자들 또한 있어 두 단축키 모두를 화면 닫기 기능으로 사용할 수 있도록 추가하여 주었다. #13262에서는 Folding 메뉴에서 Fold All의 위치가 다른 항목들과 매치가 되지 않는다고 보고하였다. 코드 반영은 이동(Move)가 되었는데 개발자들은 Fold All의 항목을 Folding 메뉴 내부의 구분선 위쪽으로 이동시켜 주었다. 요구사항 수정에 있어서는 대부분의 코드 반영의 타입이 다양한 것을 볼 수 있다.
- 세분화(Refine): Issue #6922에서는 업데이트 시 활성화되는 아이콘이 업데이트가 끝나도 여전히 활성화되어 릴리즈 노트를 가르치는 문제를 지적하였다. 개발자들은 원래 릴리즈 노트를 가리킬 때 아이콘을 회색으로 수정하기로 합의하였었지만, 해당 아이콘 자체가 없어지면서 수정이 무효화 되었다. 따라서 시스템 반영에서 코드 변경 계획은 무효화(Invalidate)되었다. #1690에서는 파일 내용이 변경될 때마다 자동으로 저장이 되도록 기능을 변경해 달라고 요청하였다. 개발자들은 요구처럼 변경할 경우 메모리 과부하, 기존 사용자들의 불편을 일으킬 수 있다고 하였다. 토론 끝에 활성화, 비활성화가 가능하도록 기능을 대체하는 쪽으로 마무리되었다. 따라서 코드 변경은 대체(Replace)타입이 되었다.
- 대체(Replace) #6351에서는 Bootstrap CSS를 Hand-written CSS로 변경해달라는 제안이 있었다. 개발자들은 Bootstrap의 필요 없는 부분을 삭제하고, Hand-written CSS를 추가하였으며, Bootstrap 일부를 core로 이동시켰고 기본 스타일과 병합시켰다. 여러 번의 패치를 통해 다수의 변경이 있었다. 따라서 시스템 반영에는 Add, Delete, Move, Merge의 코드 변경이 혼합되었다.

[1] ATOM, https://github.com/atom/atom

[표 3] ATOM 에디터 프로젝트의 요구사항 변경 요청에 대한 시스템 반영 타입 분석표

요구사항변경			시스템 반영	
ID	타입	설명	타입	설명
3111	Create	모든 탭을 한 번에 닫기 위해 CTRL+SHIFT+W 추가 제안	Add	패치를 통해 메뉴바에 모든탭 한번에 닫기 버튼를 추가
7332	Create	화면 크기를 마우스가 아닌 키보드 단축키를 통해 조절할 수 있도록 추가	Add	패치를 통해 화면 크기 조절관련 단축키를 추가
8939	Create	Top 과 Bottom bar 를 에디터에 추가하기를 제안	Add, Modify	기본 에디터 틀 변경 후, top 과 bottom bar 를 나타내는 요소 추가
960	Create	스타일과 기능이 모든 OS 에 기본이 되는 Native OS 가 필요	Add, Modify, Merge	편집기 맨 위에 전역 메뉴를 추가하고 메뉴의 항목들을 수정, 병합
6134	Create	'디렉토리 내 검색'이 여러 개의 루트 폴더가 있을 경우에도 실행될 수 있게 제안	Replace	새로운 라이브러리로 대체하여 Scandal 에서 패턴이 파일인지 디렉토리인지를 판단 할 수 있도록 * 문자를 포함하지 않는 경로 패턴을 명시
3064	Create	리눅스 상에서 OPenFolder 기능이 단축키로 존재하지 않는데 추가 제안	Replace, Add	CTRL+SHIFT+O 를 OpenFolder 로 대체하고 원래의 devOpenFolder 단축키를 새로 추가
11486	Modify	현재 CTRL+W 로 작동하는 화면 닫기 기능을 CTRL+F4 로 변경	Add	현재 CTRL+W 기능을 사용하는 유저들이 있기 때문에 CTRL+F4 기능을 추가
12951	Modify	드보락 키보드를 사용할시 쿼티 자판처럼 인식하는 문제를 수정	Add	드보드락 사용자를 위한 키맵 패치를 추가
21	Modify	수정된 buffer 만을 저장하도록 변경을 제안	Modify	수정된 buffer 만을 저장하도록 수정
12451	Modify	들여쓰기 안내선을 표시하지 않는 기존 편집기(즉, 기존 프로젝트를 열 때 자동으로 열리는 편집기)에서 안내선을 표시할 수 있도록 수정	Modify	텍스트에디터를 deserialize 한 후에 모든 매개 변수로 디스플레이 레이어를 재설정하도록 수정
13262	Modify	Folding 메뉴에서 Fold All 의 항목의 위치를 다른 항목과 매치되게 변경 제안	Move	Fold All 의 항목을 Folding 메뉴 내부의 구분선 위쪽으로 이동
7145	Modify	Dialogue 를 저장할 때 필터 유형을 포함하는 것으로 변경 제안	Refine	이전 동작을 유지하고 필터 유형을 포함하는 개선된 기능을 추가
11504	Modify	Atom 관련 파일의 아이콘을 확장자에 따라 다르게 변경 제안	Refine	기본 파일의 아이콘은 그대로 유지하고 다른 확장자의 파일의 아이콘을 다르게 개선하여 추가
6917	Modify	다른 프로젝트 폴더 추가 시, 변경 사항을 무시하고 이전 프로젝트 폴더로 돌아가는 경우가 발생하지 않도록 수정	Refine	기존 창의 경로를 변경 후 창을 종료하였을 때, 변경 사항이 저장되도록 하는 코드를 추가
7178	Modify	팬 이동 커서의 일관성을 유지하도록 변경	Replace	모양이 다른 커서를 같은 모양으로 대체
6922	Refine	업데이트 시에 활성화되는 아이콘이 끝나도 여전히 활성화되어 릴리즈 노트를 가리킴	Invalidate	원래는 릴리즈 노트를 가리킬 때 회색으로 수정하기로 합의되었으나, 해당 아이콘 자체가 없어지면서 수정이 무효화
3174	Refine	자동 quick jump 기능을 개선하여 그것을 해제하는 옵션을 추가 요청	Refine	quick jump 기능을 그대로 두고 그것을 해제할 수 있는 옵션을 추가
1690	Refine	파일 내용이 변경될 때 마다 자동으로 저장이 되도록 기능을 변경 제안	Replace	활성화/비활성화가 가능하도록 기능을 대체
6351	Replace	Bootstrap CSS 를 hand-written CSS 로 변경 제안	Add, Delete, Move, Merge	Bootstrap 필요 없는 부분 삭제, hand-written CSS 추가, Bootstrap 의 일부 Core 로 이동, 기본 스타일과 병합
12961	Replace	Native menu 를 HTML5 버전으로 변경 제안	Reject	은 기본 취지가 웹 애플리케이션의 느낌을 없애자는 철학에서 시작되었으므로, 해당 제안이 기본 철학에 위배되어 거절

우리가 조사한 요구사항 요청이 이슈 게시판에 올려져 있는 사용자들의 요구사항이다 보니 대부분이 수정(Modify)이나 생성(Create)으로 분류되었다. 그에 대한 시스템 반영은 추가(Add), 수정(Modify), 세분화(Refine)가 가장 많았다.

6. 요구사항의 진화를 위한 기존 연구의 한계점과 향후 연구 방향

6절은 3절-5절의 문헌 조사와 실 사례 조사를 통해 파악한 다음 질문의 논의를 정리하고자 한다: "요구사항의 진화를 위한 기존 연구의 한계점과 향후 연구 방향은 무엇인가?"

6.1 기존 연구의 한계점

우리는 3.1절과 3.2절의 자동화 연구를 통해 연구자들이 요구공학과 관련한 여러 자동화 기법과 도구들을 제시하고 있음을 확인하였다. 이 중에서 소프트웨어 요구사항의 진화와 관련된 연구를 선별하면, 다음과 같다.

- 요구사항 수집: 요구사항 자동 도출 기법[10]
- 요구사항 관리: 요구 변경 ID 관리 기법[9]
- 요구사항 관리: 변경 전파 시스템 [18]
- 요구사항 관리: 추적성 링크 유추 도구 [19]
- 요구사항 관리: 신뢰할 수 있는 링크 구분 [20]
- 요구사항 관리: 진화를 위한 시각화 도구 [21]
- 요구사항 관리: 요구사항과 구현물 맵핑 도구[8]

우리는 상기 요구사항 관련 자동화 기술들로 4절에서 분석한 문제점을 해결하여 요구사항의 진화 작업을 체계화 할 수 있는지 검토하였다.

첫째, 요구사항 수집의 자동화 기법[10]은 그림 1을 기준으로 보았을 때, 변경 요청에 있어 사용자의 요청사항의 지속적인 피드백 부분에 초점을 두고 있다. 나머지 요구사항 수집을 어떻게 할 것인지 지원하지 않는다.

둘째, 요구사항 관리와 관련한 자동화 기법[9]은 표 1의 추적성 매트릭스의 처음 진입점으로서의 요구사항 변경에 대한 ID 변경 관리 기법을 제시하고 있다. 이는 그림 1의 요구 사항에서의 수준에 대한 축을 고려하고 있지 않다. 또한, 4.3절에 지적한 임의의 항목에서 변경이 발생할 때의 추적성 관리의 이슈를 다루지 못한다.

셋째, 요구사항 관리에 있어서 요구사항과 구현물의 맵핑 기법[8]은 표 1의 추적성 매트릭스 형식을 따르지 않고, 구현물을 클릭하여 요구 사항과 바로 맵핑가능한 지원 도구를 제공한다는 점에서 혁신적이다. 단, 요구사항이 단순한 웹 애플리케이션에 한정하여 적용된다는 점에 한계가 있다.

기타, 요구사항 관리에 있어서 연구 [18], [19], [20], [21]은 변경 전파, 추적성 유추, 진화의 가시화 등 흥미로운 아이디어를 제시하고 있다. 그러나 4절에서 논의한 소프트웨어 요구사항의 진화에 대한 다차원적인 변경의 영향에 대한 추적을 지원하는 기법이라고 보기는 어렵다.

6.2 향후 연구 기대 방향

우리는 4절에서 논의한 어려움을 해결하는 해법을 찾기 위한 연구 방향을 3절의 문헌 조사와 5절의 실 현상 조사를 바탕으로 다음과 같이 제안한다.

- 요구 변경의 다차원 관리 도구 제안: 4.1절에서 밝힌 바와 같이 요구 변경은 여러 측면과 관련되어 있다. 이러한 측면의 맵핑과 분석을 지원하는 도구가 있다면 요구사항의 진화에 대한 체계적인 분석 및 관리가 가능할 것으로 기대한다. 예를 들어, 논문 [8]의 요구사항과 구현물 맵핑 도구의 아이디어를 확장하여 다차원의 항목들을 맵핑하기 위한 도구를 제안, 개발할 수 있을 것으로 예견한다.
- 요구 변경의 발생 처리에 대한 메커니즘 제안: 요구사항의 변경이 일반 항목의 변경과 동일한 형태라는 이해 하에 변경 발생 처리에 대한 변경 관리 메커니즘을 기대할 수 있다. 특히 5절에서의 오픈 소스를 통해 사용자 요청 사항의 타입과 시스템 반영 타입을 링크를 시키는 실증적 연구가 이러한 방향에 도움을 줄 것으로 기대한다. 이러한 메커니즘은 요구 사항의 변경에 대한 이력 관리를 제공하며, 또한 추가로 코드의 변경과 연계를 제공해 주어야 할 것으로 기대한다.
- 발생한 변경의 전파 처리에 대한 제안: 요구사항의 변경은 단순하게 소프트웨어 명세서의 요구사항을 변경할 때 그 뒤의 후행 항목에 대한 변경 영향을 파악하는 것이 아니다. 요구사항의 변경에 영향을 미칠 항목의 변경과 요구사항 변경에 영향을 받은 항목의 변경에 대한 처리를 수행해야 한다. 이러한 방향으로 순방향 그리고 역방향 변경 전파 자동화 프로세스 제안할 때 요구사항 변경과 관련한 자동화의 기반을 만들 수 있으리라 예견한다.

7. 결론

우리는 문헌 조사의 결과를 통해 다음과 같은 요구사항의 추적성의 어려움의 이유를 도출하였다. 첫째, 요구사항의 변경과 관련하여 다양한 측면이 존재한다. 둘째, 요구사항의 변경은 일반 항목의 변경과 유사하게 생성, 변경(상세화, 분할, 통합, 대체 등), 삭제와 같은 타입과 생명주기를 가진다. 셋째, 변경은 소프트웨어 산출물 어디에서든 발생하므로, 변경의 순방향 전파와 역방향 전파를 할 수 있어야 한다. 또한 우리는 실증 조사로 오픈 소스 프로젝트에서의 사용자 요청 사항에 따른 시스템 반영에 대한 타입을 명시하고 맵핑하여 요구사항 변경에 대한 시스템 변경 대응이 어떻게 발생하는지 이해하였다. 본 논문의 결과로 요구사항 진화의 어려움의 요인을 규명하고 실질 이슈를 이해하는데 도움을 받아, 향후 요구사항 진화 관련 자동화 연구에 기여하기를 기대한다.

8. 참고 문헌

[1] Patrick Rempel, Patrick Mäder, Tobias Kuschke, and Jane Cleland-Huang. "Mind the gap: assessing the conformance of software traceability to relevant guidelines." Proc. 36th Int. Conf. on Software Eng. (ICSE 2014). ACM, New York, NY, USA, 943-954. 2014.

[2] Jane Cleland-Huang, Orlena C. Z. Gotel, Jane Huffman Hayes, Patrick Mäder, and Andrea Zisman. "Software traceability: trends and future directions" Proc. of the on Future of Software Eng. (FOSE 2014). ACM, New York, NY, USA, 55-69. 2014.

[3] Hazeline U. Asuncion, Frédéric François, and Richard N. Taylor. "An end-to-end industrial software traceability tool" Proc. of the the 6th joint meeting of the European Software Eng. Conf. and the ACM SIGSOFT symposium on The foundations of software eng. (ESEC-FSE '07). ACM, 115-124. 2007.

[4] Mäder, Patrick, and Alexander Egyed. "Do developers benefit from requirements traceability when evolving and maintaining a software system?" Empirical Software Eng. 20.2, 413–441. 2015.

[5] Lehman, Meir M. "Programs, life cycles, and laws of software evolution." Proc. of the IEEE. 68.9, 1060-1076. 1980.

[6] Pandey, Dhirendra, Ugrasen Suman, and A. K. Ramani. "An effective requirement engineering process model for software development and requirements management." Advances in Recent Technologies in Communication and Computing (ARTCom), 2010 Int. Conf. on. IEEE. 2010.

[7] Sommerville, Ian. "Software Engineering. International computer science series." Ed.9 Addison Wesley. 2011.

[8] Garcia, Jorge Esparteiro, and Ana CR Paiva. "A Requirements-to-Implementation Mapping Tool for Requirements Traceability." J. of Software. 193-200. 2016

[9] 김주영, 류성열, 황만수. "추적테이블을 이용한 요구사항 변경관리 및 추적 효과 연구." 정보처리학회논문지. D 17.4, 271-282. 2010.

[10] Galvis Carreño, Laura V., and Kristina Winbladh. "Analysis of user comments: an approach for software requirements evolution." Proc. 2013 Int. Conf. on Software Eng. (ICSE '13). IEEE Press, Piscataway, NJ, USA, 582-591. 2013.

[11] 장종인, 백종문. "토픽 모델링과 이해관계자 요구 산출물을이용한 요구사항 자동 우선순위화." 정보과학회논문지. 43.2, 196-203. 2016.

[12] 고덕윤, 박수용, 김순태, 유희경, 황만수. "요구사항 시나리오 기계 학습을 이용한 자동 소프트웨어 요구사항 패턴 추출 기법." 한국인터넷방송통신학회 논문지. 16.1, 263-271. 2016.

[13] 김찬회, 김종배. "요구사항 추적모델 개선 연구." 한국디지털콘텐츠학회논문지. 13.2, 247-254. 2012.

[14] 엄석환, 강성원, 김진규, 이선아. "소프트웨어 공학: 소프트웨어 제품 라인의 요구사항과 아키텍처 간 추적성 모델링." 정보처리학회논문지. 소프트웨어 및 데이터 공학 4.11, 487-498. 2015.

[15] PSEG, 요구사항 관리 도구, http://pseg.or.kr/pseg/casereq, (2016 년 12 월 접근)

[16] Yianna Papadakis Kantos, “IBM Rational DOORS: Linking and traceability,” https://www.youtube.com/watch?v=2tN_cVQP214&list=PLFB5C518530CFEC93, Apr 2012.

[17] Gotel, Orlena CZ, and C. W. Finkelstein. "An analysis of the requirements traceability problem." Requirements Eng., 1994., Proc. 1st Int. Conf. on. IEEE, 1994.

[18] Cleland-Huang, Jane, Carl K. Chang, and Mark Christensen. "Event-based traceability for managing evolutionary change." IEEE Trans. on Software Eng. 29.9, 796-810. 2003.

[19] Samuel Klock, Malcom Gethers, Bogdan Dit, and Denys Poshyvanyk. "Traceclipse: an eclipse plug-in for traceability link recovery and management." Proc. 6th Int. Workshop on Traceability in Emerging Forms of Software Eng. (TEFSE '11). ACM, 24-30. 2011.

[20] Cleland-Huang, Jane, Mona Rahimi, and Patrick Mäder. "Achieving lightweight trustworthy traceability." Proc. 22nd ACM SIGSOFT Int. Symposium on Foundations of Software Eng. (FSE 2014). ACM, 849-852. 2014.

[21] Shinobu Saito, Yukako Iimura, Hirokazu Tashiro, Aaron K. Massey, and Annie I. Antón. "Visualizing the effects of requirements evolution." Proc. 38th Int. Conf. on Software Eng. Companion (ICSE '16). ACM, 152-161. 2016.

[22] Philip Achimugu, Ali Selamat, Roliana Ibrahim, and Mohd Naz’ri Mahrin " A systematic literature review of software requirements prioritization research " Inform. and Software Technology. 56.6, 568–585. 2014.

[23] Neil Ernst, Alexander Borgida, Ivan J. Jureta, and John Mylopoulos. "An overview of requirements evolution." Evolving Software Systems. 3-32. 2014.

[24] Robinson, William, and Radu Vlas. "Requirements Evolution and Project Success: An Analysis of SourceForge Projects." SYSTEMS ANALYSIS AND DESIGN (SIGSAND), AMCIS. 2015.

[25] Stephan Schneider, Jan Wollersheim, Helmut Krcmar, and Ali Sunyaev. "How do requirements evolve over time? A case study investigating the role of context and experiences in the evolution of enterprise software requirements." J. of Inform. Technology. 1-19. 2016.

[26] Asuncion, Hazeline U. "Towards practical software traceability." Companion of the 30th int. Conf. on Software eng. (ICSE Companion '08). ACM, 1023-1026. 2008.