□ 기술해설 □

통합 CASE의 기술현황 및 발전방향

숭실대학교 정기원* · 이광용**

목 차

1. 서 론

높은 품질의 소프트웨어를 경제적으로 예정된 기간내에 개발하여 사용코자 하는 노력은 소프트웨어 공학 분야의 핵심과제이다. 이러한 목표를 달성하고자 선진국에서는 많은 연구를 수행하여 왔고 그 결과 효과적인 소프트웨어 개발환경에 활용하도록 수많은 CASE 도구들이 출현하였다. CASE 도구는 1980년대 중반까지는 얼마되지 않았으나 현재에는 100여개 이상의 회사에서 세계 시장에 CASE 도구들을 판매하고 있으며 시장규모도 급속도로 커지고 있다. 현재 국내에도 Foundation(Method/1, Design/1, Install/1 등), Excellerator, IEF, IEW, SILVERUN, 등 많은 도구들이 판매되고 있으며 이러한 도구들의 사용으로 소프트웨어 개발과정에서 생산성의 증대와 품질향상에 기여하리라 판단된다[2, 10,11,12,16].

또한, 최근에는 CASE 업계가 새로운 방향으로 급속히 움직이고 있다. 지금까지의 개별적인 단일 기능 제품으로부터 소프트웨어 생명주기 전체를 취급하는 통합 CASE의 새로운 세대로 들어가고 있음이 분명하다. 이 세대가 되면 CASE 사용자들은 개발 여건에 따라 필요로하는 프로세스와 도구들을 마음대로 재조합해서 사용할 수 있게 될 것이다[1,4,5,9,15,16].

그러나, 이러한 외국 소프트웨어 공학도구들의 지속적인 도입은 막대한 외화 지출을 감내하여야할 뿐만아니라 국내 환경에 잘 적응시켜 활용하도록 하여야 하는 추가적인 노력 부담을 안고 있다. 한글/한자 사용을 밑바탕으로 하는 화면 설계나 보고서 양식 및 기재요령 등 한국적 문화에 걸맞는 CASE 도구의 개발이 절실한 실정이다. 최근 국내의 대학, 연구소 및 일부 기업에서 CASE 도구 개발을 위한 노력을 수행하고 있어 그 성과가 일부 나타나고 있음은 다행한 일이나 아직 깊이 있는 연구와 경쟁적 개발에 이르지 못하고 있는 실정이다.

이에, 본 고에서는 현재의 CASE 기술현황 및 발전현황, 통합 CASE의 기본구조와 기능, 그리고

*종신회원
** 준회원

미래의 통합 CASE의 발전방향 등에 대해 살펴보기로 하겠다.

2. CASE의 기술현황

2.1 CASE의 발전

응용 분야 개발을 지원하기 위해 설계된 도구들과 방법론들은 오늘날의 CASE의 환경으로 서서히 발전하여 왔다[4,5,17]. 소프트웨어 위기의 발생과 이를 해결하기 위한 노력들은 많은 방법론들과 이에 도움을 주는 도구들을 탄생시켰다. 또한, 이러한 방법론들과 도구들은 개발하기 위한 응용분야 특성에 촛점을 맞추어 제안되었으며, 이들은 그림 1과 같이 서로 밀접한 관계를 가지고 있음을 알 수 있다. 즉, 특정 응용분야의 필요성에 의해 방법론들이 개발되었으며, 또한 그 방법론에 적합한 도구들을 만들게 되었다.

응용분야, 방법론, 도구들의 이러한 발전형태를 살펴보면 표 1과 같다. 먼저, 응용분야에 있어 1970년대는 대부분의 업무는 제3세대 언어를 이용한 일괄처리 시스템이었으며, 데이타베이스 기술이 성숙되면서 더욱 복잡한 시스템이 요구되었고, 자료중심의 온라인 처리 시스템이 개발되었다. 또한, 70년대 말에 접어들면서는 의사결정지원시스템(Decision Support System)이 다수 등장하게 되었다. 1980년대에는 제어와 통신 장치에 내장되는 실시간 소프트웨어 개발이 본격화되기 시작했으며, Ada와 같은 언어들이 탄생되었다. 80년대 중반에는 전문가 시스템과 지식기반 응용들에 많은 관심을 가지게 되었으며, 80년대 말에는 전략 정보 시스템이 등장하였고, 워크스테이션을 통해서 내외 정보를 추출해내는 정보시스템을 정보시스템을 사용하기 시작했다. 한편, 1990년대 이후에는 70년대와 80년대에 그 유용성을 입증받은 여러 응용 시스템들의 기능들을 하나로 통합하여 사용자들이 언제, 어느 곳에서든지 편리하게 정보를 공유하도록 하는 형태로 시스템이 발전할 것으로 예측하고 있다.

방법론 분야에 있어서도 응용분야의 개발을 쉽게하기 위해 여러가지 방법들이 탄생하게 되

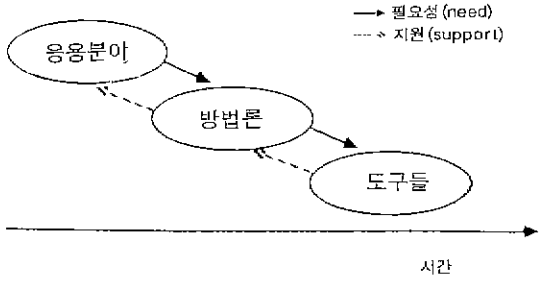

그림 1 응용분야, 방법론, 도구들 사이의 관계

표 1 CASE 도구의 발전

	1970	1980	1990
응용분야	* 일괄처리 시스템 * 온라인처리 시스템 * 의사결정 시스템	* 실시간 시스템 * 전문가 시스템 * 전략정보 시스템	* 멀티미디어 시스템 * 통합시스템
방법론	* 구조적 프로그래밍 * 구조적 설계 * 구조적 분석 * 정보시스템 설계	* 소프트웨어 메트릭스 * 프로토타이핑 * 정보공학 * 형식방법론 * 객체지향 방법	* 소프트웨어 프로세스 관리 * 통합 방법론들
도구	* 고수준 컴파일러 * 대화식 프로그래밍 * 구조, 텍스트 기반 CASE * 코드 생성기	* 4세대 언어 * 구조적 및 그래픽 기반 CASE * 통합된 프로젝트 지원 환경(IPSE) * 정보공학 기반 CASE * 역공학 CASE * CASE 쉘	* 정보저장소 기반 CASE * 지능형 CASE * 상호협력 CASE * 통합 CASE

었다. 70년대에는 자료중심의 온라인처리 시스템 개발을 지원하기 위해 체계적인 방법으로 구조적 프로그래밍 기법과 오류검출비용의 절감을 위한 구조적 분석/설계 기법이 차례로 소개되었다. 80년대 중반 이후에는 구조적 기법의 단점을 보완하는 새로운 실시간 시스템 설계기법, 객체지향 분석/설계 기법들이 제시되었고, 또한 이 시기에는 전체 생명주기를 고려한 소프트웨어 생명주기에 대한 연구도 많은 발전을 엿볼 수 있다. 이 밖에도 사용자와 개발자가 공동으로 시스템 개발에 참여하여 도움을 주는 방법론도 있었고, 70년대에 대두 되었던 구조적 기법의 기호적 표기법의 보강과 사용자 중심의 기법으로 발전되었다. 그러나, 지금까지의 방법론을 지원하는 CASE 도구들은 대부분 특정한 방법론의 특정 부분에 해당하는 기능들만을 지원하는 도구들로 국한되어 있는 형편이다.

현재까지 발전되어온 CASE 도구 기술은 다음과 같이 3세대로 분류할 수 있다. 1세대인 70년대 초반에는 CASE 도구들은 대부분이 작업본체와 문서기반이었다. 이러한 도구들은 70년대에 대두된 구조적 방법론을 지원하는 도구들이었다. 그러나, 이 도구들에 의해 저장되는 정보들은 그 자체 도구에서만 이용가능하였고, 서로 다른 도구 사이에는 정보 공유가 불가능하였다. 2세대인 80년대 초반에는 CASE 도구들은 주로 도식적 효과를 이용하여 구조적 방법론을 지원하는 형태였다. 이러한 도구들이 저장하고 있는 상세한 개발 정보는 같은 환경에서 존재하는 다른 CASE 도구들과 공유할 수 있도록 과제사전(project dictionary)에 저장되었다. 그러나, 이러한 공유도 제한적이어서 같은 회사 제품내에서만 공유가 가능하였다. 80년대 후반에 들어서서 비로서 자료저장소(repository) 기반 CASE 상품이 출현하였으며, 이러한 CASE 상품의 자료저장소에는 계획, 분석, 설계, 코딩, 시험 그리고 유지보수를 위한 소프트웨어 부품들에 대한 정보를 통합하여 관리하고 있다. 그러나, 이 CASE 도구들도 특정 응용분야 개발을 위한 특정 방법론에 지극히 의존하고 있는 한계를 가지고 있다. 3세대인 1990년대 이후에는 응용분야 및 방법론의 발전 추세로 보아 개방형 시스템을 이용한 통합 CASE 환경으로 발전될 것으로 전망된다. 이 통합 CASE 환경은 도구들간의 개발정보를 교환할 수 있도록 개발되어져야 하고, 미래의 새로운 방법론들에 쉽게 적응할 수 있도록 개발되어져야 하며, 다른 도구들에 이식성이 있도록 설계되어져야 할 것이다.

이와같이, CASE 도구들의 발전은 응용분야, 방법론 분야의 필요성에 의해 함께 발전되어 왔다. 이러한 추세로 볼 때, CASE 도구는 짧은 발전역사에도 불구하고 사용자들로부터 긍정적인 반응을 불러일으킬 것으로 판단되며, 1990년대의 응용분야와 방법론 분야의 개방화, 분산화, 통합화 등의 성격에 맞추어 CASE 도구 분야에 있어서도 이러한 추세에 따를 것으로 본다.

2.2 CASE 도구의 종류

소프트웨어 개발 환경은 소프트웨어 개발(Software Development)이라는 입장에서 언어중심환경, 구조중심환경, 툴킷환경, 방법기반환경으로 소프트웨어를 개발하기 위한 모든 도구 및 지원환경들로 분류할 수 있다. 이것을 CASE라는 측면에서, 즉, 컴퓨터의 자동화된 지원(Computer-Aided)과 소프트웨어 공학(Software-Engineering)이라는 측면에서 다시 분류하면 CASE 도구들은 툴킷(Toolkit) 기반 CASE 환경과 방법론(Methodology)기반 CASE 환경의 두종류로 분류할 수 있다[16,18].

먼저, 툴킷 기반 CASE환경은 소프트웨어 생명주기의 특정한 단계나 시스템의 한 유형의 개발 자동화에 중점을 둔 단계수준 CASE 도구(phase level CASE tool)들의 집합을 의미한다. 이러한 CASE 도구들로는 분석/설계 툴킷(analysis/design toolkit), 프로그래밍 툴킷트(programming toolkit), 통합/테스팅 툴킷(integration/testing toolkit), 유지보수 툴킷(maintenance toolkit), 프로젝트 관리 툴킷(project management toolkit), 구조화 툴킷(framework toolkit), 지원툴킷(support toolkit) 등이 있다.

분석/설계 툴킷은 시스템 요구사항을 기술하고, 그 분석결과를 설계에 이용하기 위하여 자동화한 도구들로 구성되어 있다. 여기에는 구조

적 분석/설계(SA/SD) 도구, 프로타이핑/시뮬레이션도구, 사용자 인터페이스 설계/개발 도구, 분석/설계 엔진 등 다양하다.

프로그래밍 툴킷은 프로그램 구현을 지원하는 도구들로 컴파일러, 편집기, 디버거 등과 같은 프로그래밍 언어를 중심으로한 환경 모두를 포함한다. 또한, 객체지향 프로그래밍 환경, 4세대 언어들, 응용 프로그램 생성기들, 데이타베이스 질의어들도 이 범주에 속한다. 프로그래밍 툴킷들을 예를들면 Unix의 프로그래머 워크벤치(Unix/PWB), DEC VMS VAX-set, Interlisp 프로그래밍 환경, Ada 프로그래밍 지원 환경(APSE) 등등이 있다.

통합/테스팅 툴킷은 원시코드의 오류를 검사하는 도구들이다. 이 툴킷에는 정적 분석 도구(Static Analysis Tools), 동적 분석 도구(Dynamic Analysis Tools), 시험 관리 도구(Test Management Tools)들이 있다. 정적 분석 도구는 직접적인 프로그램 실행을 수행하지 않고 원시코드를 분석하고, 동적 분석 도구는 프로그램 실행동안에 원시코드를 분석한다.

유지보수 툴킷은 새로운 시스템의 구축이 아닌 기존 소프트웨어를 관리하는 도구들이다. 현재 많은 MIS 조직에서 신규 시스템을 개발하는 것보다 현존 시스템을 유지보수하는데 더 많은 시간을 소비하고 있는 것으로 보아 유지보수 툴킷은 여러 툴킷들 중에서 가장 중요한 부분을 차지한다. 이 분야의 툴킷들은 코드 재구조화(restructuring) 도구들, 역공학(reverse engineering) 도구들과 재공학(re-engineering) 도구들로 분류되고 있다.

프로젝트 관리 툴킷은 프로젝트 관리자로 하여금 소프트웨어 프로젝트의 계획(planning), 조직구성(organizing), 인사배치(staffing), 감독(directing), 통제(controlling) 등의 활동들을 지원하는 자동화된 도구들이다. 예를들면, 프로젝트 관리를 위한 이러한 여러 가지 활동들을 돕기 위해 여러 가지 추정 도구들, 시간표/일정 도구들, 품질 메트릭스(metrics)들이 이용되고 있다.

구조화 툴킷은 개개의 소프트웨어 도구들을 조합하거나 커스터마이징(customizing)하여 관리하기 위한 내부 구조를 제공하는 도구의 유형을 말한다. 다시말하면, 초기 통합된 프로젝트 지원 환경(IPSE; Integrated Project Support Environment)으로 여러 가지 CASE 도구들, 전통적인 프로그램 구현도구들, 데이타베이스 관리도구들 모두가 통합될 수 있도록 한 것이다. 이러한 구조화 툴킷의 도입으로 말미암아 전에는 호환적이지 못한 신·구 도구들과 여러 회사에서 공급하는 도구들이 서로 쉽게 사용될 수 있는 환경으로 조합될 수 있는 가능성을 보였다.

지원 툴킷은 소프트웨어 생명주기 전반에 걸쳐 공통적으로 응용되는 도구들의 그룹을 의미한다. 이 툴킷들에는 문서화 도구들, 네트워크 관리 및 통신 도구들, 품질 보증 도구들, 소프트웨어 구성관리 도구들, 소프트웨어 검증·확인 도구들 등등 여러 가지가 있다.

그 다음으로, 방법론 기반 CASE환경은 특정 소프트웨어 방법론(methodolgy)과 관련된 소프트웨어 프로젝트 관리활동은 물론 소프트웨어 시스템의 구축 및 유지보수의 프로세스를 지원하는 통합된 도구들의 집합이다.

이 방법론 기반 CASE환경은 개발 회사별로 나름대로의 개발방법론 예를들면, DeMarco의 구조적 분석, Jackson의 구조적 설계, Yourdon의 구조적 분석/설계, Martin의 정보공학, 객체지향 방법론 등을 지원하며, 또한 개발 회사 나름의 프로젝트 관리 방법을 지원하고 있다. 현재까지는 대부분의 회사들에서 거의 대부분이 개발방법론을 Martin의 정보공학과 Yourdon/DeMarco의 구조적 방법론에 그 토대를 두고 있으나 기술발전의 추세로 보아 객체지향 방법론을 적용한 CASE 환경이 확대될 것으로 보인다. 또한, 프로젝트 관리 방법은 회사별로 거의 한가지 정도의 프로젝트 관리 정책을 쓰고 있다.

이 CASE 환경에서는 자동화 대상의 프로세스 모델이나 방법이 고정되어 있는 상태에서 시스템의 구축과 지원에 필요한 기술정보와 프로젝트 관리 정보를 일원화하여 관리한다. 한 생명주기 단계의 산출물은 다음 단계에 직접적으로, 자동적으로 옮겨질 수 있도록 통합되어 있다. 그리고, 방법론 기반 CASE 환경에 따라서는 소프트웨어 개발 프로세스를 자동화 할 수 있는 것 이외에 명세서 작성과 실행 가능한 소프트웨어를 얻을

수 있도록 되어 있다. 이러한 환경에서 이용하고 있는 도구들은 그래픽 기반 시스템 요구사항과 설계 명세서의 작성, 시스템 정보 검사, 분석 및 상호참조(cross-referencing), 시스템 정보와 프로젝트 관리정보의 저장, 관리 및 보고, 시스템 프로토타입의 구축 및 시뮬레이션, 시스템 코드와 관련문서의 생성, 표준과 절차의 강화, 외부 Excelerator, 객체지향 방법론을 지원하는 BNR사의 Object Time, Object International사의 사전(dictionary) 및 데이타베이스의 접근 등과 같은 툴킷기반 환경에서 이용되었던 도구들의 조합으로 이루어져 있다.

따라서, 이러한 CASE 환경에서는 기본적으로 그래픽 기능, 에러검사, 정보저장소 관리, 통합된 도구들 제시, 프로타이핑 지원, 체계화된 방법론 지원 등의 기능들을 지원하고 있다. 이러한 모든 기능을 완전하게 다루는 제품들은 소수이지만, 특정 방법론을 중심으로 만들어진 제품의 종류는 다양하다. Martin의 정보공학 계열인 IEW, IEF, 구조적 방법론을 지원하는 Index Technology사의 OOATool, StructSoft의 TurboCASE, Excel Software의 MacDesigner 등 여러 가지가 있다.

2.3 CASE 도구의 표준화 노력

현재, 소프트웨어 개발에 있어서의 낮은 생산성 문제, 컴퓨터 기종과 모델간의 호환성 결여, 타시스템에서의 이식성 부족, 사용자에게 편리성을 강조하는 친근성의 취약점 등의 문제를 극복하는 방법으로 소프트웨어 개발관리, 개발방법, 개발도구, 시험평가, 유지보수 등의 소프트웨어 생명주기 모든 단계에 대한 지침 내지는 표준을 제정하고 활용하고 있다.

소프트웨어 공학 분야의 이런 표준화 노력은 CASE 도구에서도 예외가 될 수는 없다. 현재 미국과 유럽 등지의 국가들에서는 이미 표준화 연구그룹을 조직하여 CASE 도구의 여러측면에 대한 표준안들을 제시하고 있으며, 서로 공통된 표준을 제정하기 위해 협력하고 있는 상황이다[3]. 이들은 소프트웨어 공학 도구들간의 자료교환을 위한 공통된 자료 포맷(format) 구성, 두

표 2 표준화 연구그룹의 표준화 분야

	팀공학 및 정보저장소	제어통합	자료통합	개방성	이식성
IEEE-1175			X	X	
ANSI & ISO IRDS	X		X		
OMG	X	X	X		
PCTE	X	X	X	X	X
EIA/CDIF			X	X	
ISO SC7 WG11	X		X		
ANSI X3H6	X	X	X	X	X

x 연구분야

도구간의 자료의미 전달 메카니즘 구현, 프로젝트와 산출물 자료를 저장하기 위한 정보저장소의 구현, 도구 호출, 도구들간의 상호협력, 제어통제 및 메시지처리를 위한 공통된(common) CASE 구조 제시라는 측면으로 다각도로 노력하고 있다.

이 표준화 연구그룹들의 표준화 연구분야를 살펴보면 표 2와 같다. 현재, 이 연구그룹들은 팀공학(Team Engineering), 프리젠테이션 통합 및 제어통합, 자료통합, 도구의 개방성 및 확장성, 그리고 이식성 분야들을 표준화 연구대상으로 하고 있다.

3. 통합 CASE의 구조와 기능

CASE는 Computer-Aided와 Software-Engineering이라는 두 가지 기본 개념을 가진 도구 혹은 환경이라고 볼 수 있다[9,16]. 즉, 소프트웨어 개발과 유지보수에서 컴퓨터의 자동화된 기능을 도움받아 정해진 시간과 예산범위 내에서 우수한 품질의 소프트웨어를 생산성이 높게 개발할 수 있게하는 도구 혹은 환경으로 볼 수 있다.

따라서, 단순하게 그래픽 처리나 단순한 기능 몇가지만 가진 도구를 진정한 의미에서 CASE 도구라고 할 수 없다. 그러나, 현재의 대부분의 CASE 도구들은 특정 방법론의 일부 기능을 지원하는 단순 그래픽 지원 도구 혹은 기본적인 몇가지 측정 기능이 첨가된 도구 수준에 불과하다.

이에 비해, 통합 CASE 도구는 소프트웨어 생명주기의 모든 단계에 대해 프로세스와 도구를 제공하고 이들이 유기적으로 연결되어 있으며 통합 CASE를 이용하는 사용자의 작업 진행에 맞추어서 필요로하는 프로세스와 도구들을 마음

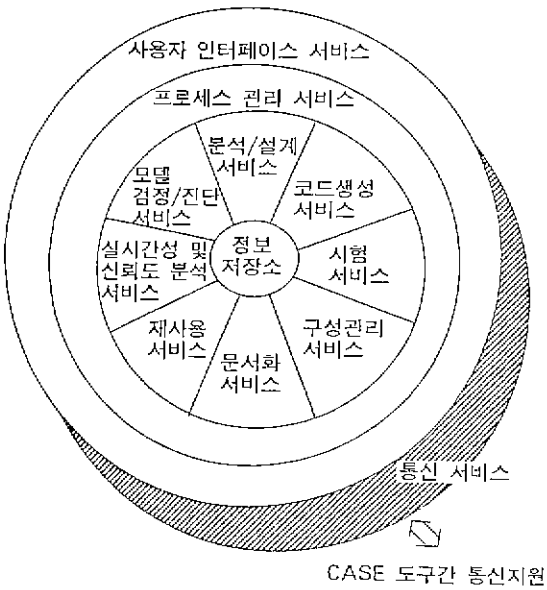

그림 2 통합 CASE의 구조

대로 재조합해서 사용할 수 있도록 적응적(adaptable)이고, 유연성(flexible)이 있고, 동적(dynamic)인 특성을 갖는다[7,8,13,14]. 특히, 최근에 CASE 제품을 생산하는 회사들이 의욕적으로 도전하고 있는 분야가 바로 자신들이 구축한 CASE 제품들을 통합해서 새로운 방법론 혹은 다른 회사 제품의 CASE 도구의 기능들을 통합해서 호환성 있게 사용할 수 있도록 하는 것이다[2].

통합 CASE를 잘 활용하게 되면 CASE 도구들간의 자연스러운 정보교환이 가능해지고, 소프트웨어 구성관리, 품질보증, 문서생성 등의 작업부담을 감소시켜 주며, 프로젝트의 통제성을 증대시켜주고, 또한 프로젝트 팀원들간의 협력체제를 증대시켜주는 등 장점이 많다. 그러나, 이러한 이득을 얻기 위해서는 통합 CASE의 기능 또한 일관성있는 표현방법 제시, 도구들간의 표준화된 인터페이스 기능제공, 다양한 하드웨어 혹은 운영체제 환경에서 수행될 수 있는 효과적인 방법제시 등 해결해야할 과제가 많다[16].

이러한 문제점들을 해결할 통합 CASE의 기본구조로 그림 2와 같은 구조를 생각할 수 있다[10,13]. 이 구조의 특징을 살펴보면, 정보저장소 서비스는 통합 CASE의 핵심부분에 위치하고 있는데, 통합 CASE 내의 모든 도구들에서 생성된 설계정보를 일관성 있게 저장하고 관리하는 핵심일 뿐만아니라 외부 CASE 환경과의 정보공유를 위한 정보기반이다. 그리고, 통신 서비스 부분은 여러도구들의 그림자 부분에 속해 있는데, 이것은 사용자가 여러 가지 통합 CASE의 내·외부 도구 사용시 발생한 메시지를 다른 도구들에 알리거나 도구들을 활성화(activate)시키기 위한 일종의 메시지 버스(message bus) 역할을 담당한 부분이다. 프로세스 관리 서비스는 프로세스 모델링 및 프로세스 관리를 지원하는 도구들을 포함하고 있어 개발 팀원들간의 협력체제를 지원하거나, 프로세스 구성요소의 통제성 확인 등의 프로세스 관련 업무를 돕는다. 이 밖에 사용자 인터페이스 서비스, 소프트웨어 개발 서비스, 지원 서비스들을 갖추고 있으며, 이들은 정보저장소 서비스와 통신 서비스를 통해 서로 유기적으로 관계를 맺고 있다.

그림 2에 정의된 서비스들은 통합(integration) 관점에서 자료통합, 제어통합, 프로세스통합, 프리젠테이션통합, 기타통합의 5가지 통합서비스로 다시 분류할 수 있다. 자료통합은 정보저장소 서비스, 제어통합 기능은 통신서비스, 프로세스 통합 기능은 프로세스관리 서비스, 프리젠테이션 통합 기능은 사용자인터페이스 서비스와 관련이 있으며, 기타 통합 기능으로 도구의 계층적인 통합(tool layer integration)과 도구 지원 통합(tool support integration)으로 분류한다. 계층적 통합 기능은 또다시 수직적 통합 기능과 수평적 통합 기능으로 분류하는데, 수직적 통합(vertical integration) 기능은 여러 생명주기 단계별로 생성된 정보의 완전성과 일관성을 보장하는 기능을 담당하며, 수평적 통합(horizontal integration) 기능은 여러가지 모델링 방법들을 이용하는 동안 설계정보의 무결성을 유지보수하는 기능을 담당한다. 그러므로, 계층적 통합은 분석/설계 서비스, 코드생성 서비스, 시험 서비스들과 관련이 있다. 한편, 지원 통합 기능은 소프트웨어 생명주기전반에 걸쳐 공통적으로 응용되는 도구들의 통합과 관계한다. 따라서, 이 기능은 구성관리 서비스, 문서화 서비스, 재사용 서비스, 실시간성 및 신뢰도 분석 서비스 등과 관련이 있다.

자료통합의 목적은 여러 도구들에서 생성된 설계정보를 공유하는 능력에 있으며, 현재 통합 CASE 기술의 주요 핵심사항으로 여겨지고 있다. 도구들을 통합하는 이유는 여러가지의 목표(target) 환경에 잘 적응이 되는 CASE 환경을 만들려는데 있다. 그러나, 사용하는 도구들마다 각기 자신의 정보저장소를 가지고 있고, 서로 호환이 되지 못한다면 목표환경에 잘 적응이 되는 CASE 환경의 구축은 불가능할 것이다. 따라서, 이 정보저장소(repository)의 통합은 여러 회사가 제공하는 CASE 도구들을 통합하는 근간이 되는 것으로 판단하고 있다. 정보저장소가 갖추어야 할 요건은 메타모델 서비스, 질의어 서비스, 뷰(view) 서비스, 자료교환 서비스, 실질적인 인터페이스 기능 등 다양하다. 도구들간의 정보를 공유하는 방법은 4가지가 있다. 첫번째는 두 도구들간의 설계정보를 직접적으로 전달하는 방법이다. 이것은 CASE 도구를 구현한 회사들마다 정보를 표현하는 포맷(format)이 다르기 때문에 이와같이 구현하는 것은 불가능하다. 두번째는 두 도구간에 설계정보를 화일 변환을 통해 전달하는 방식이다. 세번째는 개방형 시스템과 분산환경에서 도구들간의 정보 통신을 통해 공유하는 방식이다. 네번째는 정보저장소에 객체(개체)와 링크(관계)를 저장하고 관리하는 기능, 객체들의 버젼/구성관리 기능, 명명(naming) 기능, 트랜잭션 제어 기능 등의 많은 기본적인 서비스들을 두어 정보를 공유하도록 하는 방법이다.

제어통합의 목적은 도구 사용시 발생하는 사건들을 다른 도구들에게 알리거나 다른 도구들을 활성화(active)시키는 기능을 유연하게 할 수 있도록 하는데 있다. 이상적인 통합 CASE 환경에서는 모든 도구에 의해서 제공되는 모든 기능은 다른 도구에 의해서 제어될 수 있어야 한다. 다른 도구의 기능을 공유하기 위해서 필요한 기능을 다른 도구들로부터 참조해야 하는데 이를 위해 정보저장소의 다른 도구의 기능자료를 참조해야 한다. 이러한 점에서 제어통합은 자료통합 기능을 보충하고 있다. 자료통합이 자료의 표현, 전환, 저장에 관계하는 반면, 제어통합은 제어의 전이, 서비스 공유와 관계가 있다.

프로세스통합의 목적은 사용자에 의해 정의된 프로세스가 얼마나 일관성있게 제어되는가에 있다. 이것은 정의된 프로세스대로 필요한 도구들이 일관성있게 제어되고 관리되어야 함을 의미한다. 프로세스 통합을 성취하기 위해서는 프로세스 단계(process step), 프로세스 사건(process event), 프로세스 제약사항(process constraint) 부분에 대한 통합성이 필요하다. 프로세스 단계에서의 통합성은 프로세스 단계별 활동들을 지원하기 위해 적당한 도구들이 얼마나 잘 조합될 수 있는가에 의해 확인된다. 그리고, 프로세스 사건의 통합성 확인은 어떤 프로세스를 지원하기 위해서 통합 CASE에 도구들이 얼마나 잘 일치하고 있는가에 의해 판단된다. 또한, 프로세스 제약사항에서의 통합성은 그 제약사항들을 만족시키기 위해 통합 CASE내에 도구들이 얼마나 잘 협력하고 있는가로 판단된다.

프리젠테이션 통합의 목적은 CASE 사용자가 새로운 도구를 쉽게 배우도록 하면서 일관성있게 여러 도구들과 인터페이스가 가능하도록 하는데 있다. 대부분의 윈도우 기반 도구들은 윈도우 시스템 수준, 윈도우 매니저 수준, GUI 툴킷 수준, look-and-feel 지침서 수준의 4가지 통합 기능을 갖고 있다. 현재, 대부분의 통합 CASE 구조에서는 사용자 인터페이스 서비스를 위한 일반적인 표준으로 Motif를 선택하고 있다.

4. 통합 CASE의 발전방향

앞으로의 통합 CASE 환경의 목표는 개방형 시스템 환경에서 운용이 되며, 프로젝트 팀원 혹은 개발 팀원간의 협력작업 수행, 통합 CASE의 핵심인 정보저장소의 효과적인 활용, 공유정보의 재사용성 증대, 인공지능 기술의 활용을 통한 지능형 통합 CASE 구축에 있을 것이다.

앞서 살펴본 바와 같이, CASE 도구의 발전이 응용분야의 환경변화와 밀접한 관계가 있으므로, 응용분야의 1990년대 이후의 추세로 판단해 볼 때 도구(tool) 분야에서도 통합화, 개방화, 표준화 추세로 나아갈 것이 확실하다. 현재에는 정보저장소와 자료의 무결성을 취급하는 자료통합 분야와 완전한 통합성(integration)을 획득하기 위

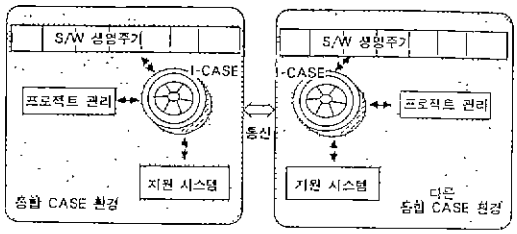

그림 3 미래의 통합 CASE 환경

한 프로세스 통합기능의 보강이 통합 CASE 연구분야의 주요 연구사항으로 취급되고 있다[1,3,4,5,8,15,17].

그림 3은 소프트웨어 생명주기 단계별로 개발팀과 프로젝트 관리팀이 통합 CASE 환경을 통한 상호작용(interaction)을 묘사한 것이다. 이 그림 3처럼 앞으로는 각각의 프로젝트팀들은 개별적으로 나름대로의 소프트웨어 생명주기 패러다임(paradigm)과 지원시스템을 가지고서 프로젝트를 수행하고 프로젝트간의 정보공유도 통합 CASE를 통해 지원받을 것이다.

현재 CASE 도구에서 적용하는 기술의 추세를 항목별로 요약하면 다음과 같다.

4.1 객체지향 방법론 적용

CASE 도구에서 수용하는 개발방법론으로는 현재까지 구조적 방법론이 주종을 이루고 있으나 정보공학론에 근거한 CASE 도구가 현재 확대되는 추세를 보이고 있다. 그러나 현재 나타나기 시작한 객체지향 방법론을 적용한 CASE 도구에 대한 관심이 높고 향후에는 객체지향 방법론이 다른 방법론들을 압도할 것으로 보인다. 그러나, 최근에 나타나기 시작한 객체지향 방법론을 적용한 CASE 도구들은 구조적인 방법론과 객체지향 방법론의 모델 표현을 지원하는 정도로 생각된다. 앞으로 객체지향 방법론을 CASE 도구에 적용하여 모델의 표현은 물론이고 일관성 및 무결성 진단과 통합 CASE 도구로서의 발전과 함께 실시간 소프트웨어 개발과 같은 시스템 소프트웨어 개발에도 직접 응용될 수 있는 도구로의 발전이 기대된다.

4.2 CASE 도구들 통합

통합 CASE 도구는 계획 수립에서부터 분석, 설계, 구현, 시험 및 유지보수 등 소프트웨어 생명주기 모든 단계에서 일관성있는 방법론에 의해 지원하는 CASE 도구를 의미한다. 이를 위해 기존의 CASE 도구들을 통합(CASE Tool Integration)함과 아울러 각 단계에서 이용하는 정보를 정보저장소(repository)에 저장하여 공동으로 이용하고자 한다. 최근 몬트리올에서 개최된 CASE '92 workshop에서는 도구 통합을 위한 여러가지 방안이 발표되었고 ECMA 표준으로 정한 PCTE (Portable Common Tool Environment)가 de facto 표준으로 도구 인터페이스에 적용되는 경향을 보이고 있다.

4.3 정보저장소(repository) 활용

통합 CASE 도구에는 각 단계를 지원하는 도구에서 생성된 정보를 저장하고 공유하는 역할을 효과적으로 지원할 수 있어야 한다. 현재 관계형 데이타베이스를 이용한 정보저장소의 구성, 자료사전의 구현, 정보저장소의 표준화, 개방시스템에의 적용 및 지능형 정보저장소 등에 관한 적용 및 연구가 활발하다.

객체지향 방법론을 적용하는 CASE 도구를 위해 객체지향 CASE 도구용 정보저장소에 관한 연구도 필요하다.

4.4 재사용 기술 활용

이미 개발된 시스템의 설계나 코드를 재사용(reuse)함은 개발 생산성과 소프트웨어 품질면에서 커다란 효과를 나타낸다. 또 기존 프로그램에서부터 설계 모델을 유도해내는 역공학(reverse engineering)과 기존 프로그램을 새로운 환경에 맞도록 변환하는 재공학(re-engineering) 등도 재사용면에서 적절히 활용할 만하다. 또한, 소프트웨어 재사용을 원활히 하기 위해서는 재사용 소프트웨어의 인식, 분류방법, 저장 및 검색기술 등이 필요하며 코드의 재사용 뿐만 아니라 상위 수준의 재사용(요구사항, 분석, 설계 등의 재사용)이 촉진되어야 한다.

4.5 프로젝트 관리 지원

CASE 도구들의 통합 CASE 도구화에 힘입어 프로젝트 관리를 지원하는 기능이 도구에 포함되는 것은 당연하다. 계획 단계에서부터 비용산정, 일정계획, 소프트웨어 프로세스 모델링 및 프로세스 관리 등과 아울러 소프트웨어 개발을 지원하는 구성관리, 품질보증, 문서화, 훈련 등의 활동을 CASE 도구는 지원하여야 한다.

4.6 실질적인 사용자 인터페이스

사용자 인터페이스를 구현함에 있어 도구의 활용이 효과적인 경우가 많다. 사용 인터페이스 툴킷(user interface toolkit)과 사용자 인터페이스 설계 지식은 훌륭한 재사용의 대상이 되며 이를 잘 활용할 수 있도록 하는 도구의 활용이 긴요하다. 또한, 그래픽 사용자 인터페스(GUI)의 응용은 윈도우의 계층적 설정과 화면에서의 모델의 표현 및 대화형 메뉴설정에 크게 도움이 되며 사용자로 하여금 개발 관리 프로세스의 각 활동을 올바르게 수행할 수 있도록 보조한다.

4.7 지능형 CASE 도구

오늘날의 소프트웨어 도구들은 기능적으로는 강력하나 생각할 줄 모른다. 단순히 사용자에게 필요한 기능만 제공하지, 사용자로 하여금 각 도구 사용의 문제점을 파악하여 스스로 해결하도록 해준다든지, 각 도구들간의 기술적 차이를 인식시켜주는 지적인 기능은 아직까지는 없다. 1990년대의 도구는 개방형 도구로서의 구조를 가지면서도 지능형 도구가 될 것이다. 이러한 지적 능력은 자연어 처리, 언어인식, 음성합성, 지식공학, 병행처리 및 프로그램 애니메이션 등과 같은 진보된 기술을 적용하므로써 가능할 것이다.

4.8 기타

CASE 도구 개발에서 개방형 도구로서의 구조, 도구 개발 기관간의 표준화로 통합 메카니즘 구축(PCTE, CDIF, CAIS 등), CASE 통합을 위한 참조모델의 구성(NIST/ECMA reference model), 형식(formal) 명세 방법 적용, 테스트 케이스 자동생성, 프로토타입 개발 및 시뮬레이션, 모델 검증 및 진단 등 많은 분야에서의 연구 및 적용 실험이 이루어지고 있어 이에 대한 관심과 연구 및 적용 노력이 필요하다.

5. 결 론

지금까지, 통합 CASE의 기술현황과 발전방향에 대해서 살펴보았다. 현재, 소프트웨어 개발에 있어서의 낮은 생산성 문제, 컴퓨터 기종과 모델간의 호환성 결여, 타시스템에의 이식성 부족, 사용자에게 편리성을 강조하는 친근성의 취약점 등 문제가 많다. 이것을 극복하기 위한 한 방편으로 CASE라는 자동화된 소프트웨어 공학 도구를 활용하여 많은 도움을 얻고 있다. 그러나, 현재의 대부분의 CASE 도구들은 특정 방법론의 일부 기능을 지원하는 단순 그래픽 지원 도구 혹은 기본적인 몇가지 측정 기능이 첨가된 도구에 불과하다. 그 결과, 최근에 CASE 제품을 생산하는 회사들에서는 진정한 의미의 CASE 도구를 구축하기 위해 의욕적으로 소프트웨어 공학의 여러 기능을 통합한 CASE 환경을 만들려고 노력하고 있다.

소프트웨어 공학자들은 1990년대 이후의 CASE는 개방형 시스템을 이용한 지능형 통합 CASE의 시대라고 예측한다. 이러한 환경하에서 소프트웨어를 개발하게 되면, 소프트웨어 개발 생산성은 물론, 유지보수, 호환성, 이식성, 친근성, 프로젝트 팀원간의 협력체제 구성, 프로젝트의 통제성 등의 소프트웨어 분야의 여러 문제들이 해결될 것으로 보인다. 그러나, 이러한 통합 CASE를 구축하기 위해서는 통합 CASE의 기능 또한 일관성있는 표현방법 제시, 도구들간의 표준화된 인터페이스 제공, 다양한 하드웨어 혹은 운영체제하에서 수행될 수 있는 효과적인 방법 제시 등등 해결해야할 과제가 많다.

앞으로, CASE 도구의 발전흐름에 맞추어 빠른 시일 내에 객체지향 개발방법론 완성, 소프트웨어 프로젝트 관리방법 완성, 각종 도구들의 자

동화, 개방형 환경 구축 등의 CASE와 관련된 기반연구들을 마치고, CASE 기능들의 통합화, 개방화, 분산화, 표준화 등의 노력을 경주해야 하겠다.

참고문헌

[1] A. W. Brown, P. H. Feiler, and K. C. Wallnau, "Past and Future Models of CASE Integration," Fifth International Workshop on Computer-Aided Software Engineering, Jul., 1992. pp. 36~45.

[2] C. Gane, Computer-Aided Software Engineering: The Methodologies, The Products, and The Future, Prentice-Hall, Inc., 1990.

[3] CASE '92 Workshop Report: Results from the Fifth International Workshop on Computer-Aided Software Engineering, Jul., 1992.

[4] CASE World: Conference Proceedings, Mar., 1993, Vol. I.

[5] CASE World: Conference Proceedings, Mar., 1993, Vol. II.

[6] I. Thomas, "PCTE Interfaces: Supporting Tools in Software-Engineering Environments," IEEE Software, Nov., 1989, pp. 15~23.

[7] I. Thomas and B. A. Nejmeh, "Definitions of Tool Integration for Environments," IEEE Software, Mar., 1992, pp. 29~35.

[8] J. Estublier, N. Belkhatir, M. Ahmed Nacer, W. L. Melo, "Process Centered SEE and Adele," Fifth International Workshop on Computer-Aided Software Engineering, Jul., 1992, pp. 156~165.

[9] K. Spurr and P. Layzell, CASE: Curent Practice, Future Prospects, John Wiley & Sons Ltd., 1992.

[10] 국방과학연구소 ATRC-409-93652, 국방 소프트웨어 개발 및 관리모델에 관한 연구, 1993. 10.

[11] 과학기술처 UC N31530, 소프트웨어 자동생산 기술에 관한 연구: 소프트웨어 개발 및 관리체계 개발에 관한 연구, 1993. 10.

[12] 과학기술처 UC M20431, 소프트웨어 자동생산 기술에 관한 연구: 요구분석 지원도구의 실용화, 1993. 10.

[13] M. Chen and R. J. Norman, "A Frameworkk for Integrated CASE," IEEE Software, Mar., 1992, pp. 18~22.

[14] P. Mi and W. Scacchi, "Process Integration in CASE Environments," IEEE Software, Mar., 1992, pp. 45~53.

[15] R. Balzer, T. E. Cheathan, C. Green, "Software Technology in the 1990's Using a New Paradigm," IEEE Computer, 1983, pp. 39~45.

[16] R. S. Pressman, Software Engineering: A Practioner's Approach, Third Ed., McGraw-Hill, Inc., 1992.

[17] R. J. Norman and M. Chen, "Working Together to Integrated CASE," Software, Mar.. 1992, pp. 12~16.

[18] S. A. Dart, R. J. Ellison, P. H. Feiler, and A. N. Habermann, "Software Development Environments," IEEE Computer, Nov., 1987, pp. 18~28.

정 기 원

1967 서울대학교 전기공학과 졸업
1982 미국 알라바마 주립대학(헌츠빌) 전산학 석사
1983 미국 텍사스 주립대학(알링턴) 전산학 박사
1966 ~1968 미필군 전자기사
1969 ~1971 대한전자공업 주식회사 (자료처리과장)
1971 ~1975 한국과학기술연구소 전자계산실
1975 ~1990 국방과학연구소 책임연구원
1990 ~현재 숭실대학교(소프트웨어공학과) 부교수
관심 분야 : 소프트웨어공학, 모델링/시뮬레이션, 실시간시스템, 분산처리, 인공지능

이 광 용

1991 숭실대학교 전자계산학과(학사)
1993 숭실대학교 대학원 전자계산학과(석사)
1993 ~현재 숭실대학교 대학원 전자계산학과 박사과정
관심 분야 : 소프트웨어공학, 실시간시스템, 인공지능 모델링/시뮬레이션