메트릭 기반 프로젝트 관리를 위한 방법 및 도구 개발

신현일*, 최호진*, 백종문*
*한국정보통신대학교 공학부
e-mail : {linugee, hjchoi, jbaik}@icu.ac.kr

Development of Methods and Tools for Metrics-Based Project Management

Hyunil Shin*, Hojin Choi*, Jongmoon Baik*
*Information and Communications University, School of Engineering

요 약

체계적이고 지속적인 소프트웨어 개발 프로세스 측정 및 분석 활동은 프로젝트 관리에 있어서 중요한 요소 중의 하나로 알려져 있다. 하지만 측정 및 분석 활동에 요구되는 높은 비용과 신뢰성 있는 메트릭 데이터 획득의 어려움으로 인해 측정 및 분석 활동을 수행하는 데에는 많은 어려움이 존재한다. 본 논문에서는 메트릭 자동 수집 및 분석 도구와 이 도구를 기반하여 소프트웨어 프로젝트 관련 문제 인식과 문제 해결을 할 수 있게 하는 방법을 제안한다. 도구와 방법을 통해 메트릭 수집에 대한 오버헤드를 없애 측정 및 분석 활동의 수행을 쉽게 할 수 있고 수집된 메트릭의 분석을 통해 객관적으로 이슈를 파악하고 해결 할 수 있다. 또한 프로젝트 중에 메트릭의 지속적인 수집이 가능하고 수집과 동시에 수집된 메트릭에 대한 분석을 할 수 있기 때문에 잠재적인 위험 요소의 조기 식별과 해결을 가능케 하여 소프트웨어 제품의 품질향상과 문제해결에 드는 비용의 절감 효과를 얻을 수 있다.

1. 서론

소프트웨어 공학적 연구 측면에서 메트릭이 프로젝트 관리에 있어서 중요한 역할을 한다는 것은 널리 알려져 있다. “측정할 수 없는 것은 예측할 수도 제어할 수도 없다[1].”라는 말이 있듯이 소프트웨어 메트릭의 일관된 수집과 분석은 소프트웨어 제품과 개발 프로세스를 객관적인 방법으로 관리할 수 있게 한다. 효과적인 측정 및 분석 활동은 소프트웨어 개발 조직이 자신의 개발 역량을 이해할 수 있도록 도와주고 이러한 평가를 바탕으로 소프트웨어 제품 개발과 배포에 있어서 달성 가능한 계획을 세울 수 있게 해 준다. 프로젝트 관리 관점에서 측정과 분석 활동을 통한 객관적인 정보의 획득으로 인해 얻을 수 있는 이득에는 효과적인 의사소통, 프로젝트 목적의 추적, 문제의 조기 식별 및 대처, 대안절충 의사결정, 의사결정의 정당화가 있다 [2]. 이 외의 이득으로는 CMMi™ (Capability Maturity Model Integration)[3], PSP (Personal Software Process)[4], SPICE (Software Process Improvement and Capability dEtermination)[5]가 메트릭 측정과 분석 활동의 필요성을 강조하고 있듯이 소프트웨어 프로세스 개선의 측면에서도 측정과 분석은 매우 중요한 요소로 자리잡고 있다.

그러나 소프트웨어 측정과 분석의 중요성이 인식되고 실현될 경우 이전에 언급한 이익들을 획득할 수 있음에도 불구하고 실제로 효과적으로 메트릭의 수집과 분석 활동을 수행하는 소프트웨어 개발 조직은 그리 많지 않다. 메트릭 수집과 분석 활동에 요구되는 높은 비용이 그 주된 요인이다. 기존의 측정 방법론과 도구들은 개발자 또는 메트릭 측정을 담당하는 이들이 메트릭을 기록하는 것을 요구한다. 이러한 방법은 개발자에게 많은 오버헤드를 부과시키거나 측정을 담당하는 이를 고용하기 위한 추가 비용을 요구한다. 또한 수집되는 데이터의 신뢰성과 관련하여 개발자나 측정의 담당자가 수집, 기록하는 메트릭 데이터는 수용 가능한 신뢰도를 제공하기에 어려움을 가지고 있

본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT 연구센터 육성, 지원사업의 연구결과로 수행되었음

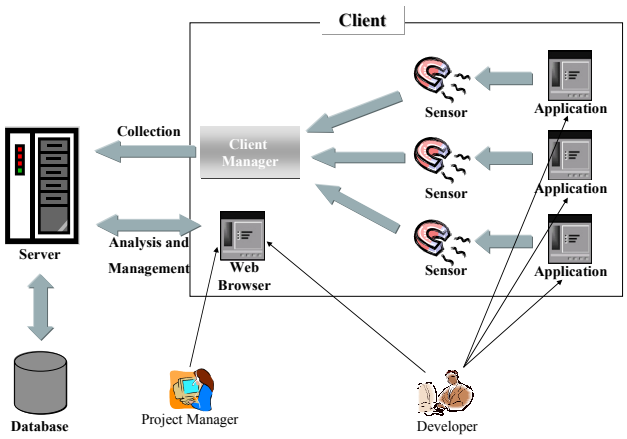
다. 예를 들어 개발 업무에 충분히 바쁜 개발자에게 신뢰도 있는 측정을 요구하는 것은 무리가 될 수 있다.

따라서 본 논문에서는 개발자의 개입이 필요 없이 자동으로 메트릭 데이터를 수집해 주는 도구와 이 도구를 통해서 프로젝트 관리 시 객관적으로 문제를 인식하고 파악된 문제를 해결하도록 도와주는 방법을 제안한다.

2 장에서는 메트릭 자동 수집 및 분석 도구를, 3 장에서는 이 도구를 기반으로 하여 소프트웨어 프로젝트 관련 문제 인식과 문제 해결을 할 수 있도록 하는 방법을 제안한다. 4 장에서는 관련 연구를 소개하고 5 장에서는 결론 및 향후 연구에 대해서 기술한다.

2. 메트릭 자동 수집 및 분석 도구

이 장에서는 개발자의 개입이 필요 없이 자동으로 메트릭을 수집하고 수집된 데이터를 분석하는 도구를 기술한다. 메트릭의 자동 수집과 분석 도구를 위한 모듈 간의 관계를 나타내면 (그림 1)과 같다.



(그림 1) 측정 및 분석 도구

2.1 센서(Sensor)

센서는 메트릭을 자동으로 수집해 주는 소프트웨어 도구로써 시간, 크기, 품질과 복잡도 등의 메트릭을 수집한다. 센서에는 두 가지 종류가 있다. 하나는 개발 도구(예, Eclipse, Visual C++)에 플러그인 형태로 삽입 되어 메트릭을 수집하는 센서이고, 다른 하나는 운영체제의 시스템 호출을 감시하여 메트릭을 수집하는 센서이다. 플러그인 센서는 코딩, 테스팅, 설계, 요구사항, 형상관리 도구 등에 삽입되어 필요한 메트릭을 자동으로 수집한다. 수집된 메트릭은 클라이언트 매니저로 전송된다. 센서의 핵심은 개발자에게 자신의 주요 업무인 소프트웨어 개발 활동으로부터 메트릭 수집 활동으로 인한 오버헤드를 완전히 없애준다는 데 있다. 그럼으로써 신뢰성 있고 일관된 메트릭 수집을 가능하게 한다. 프로젝트에서 사용될 주요 센서들의 종류는 다음과 같다.

- TimeSensor: 시스템 호출 감시 센서. 애플리케이션 사용 시간을 수집한다.
- FileMonitor: 시스템 호출 감시 센서. 개발자가 수정 중인 파일에 대한 정보를 수집한다.
- CVS (Concurrent Versions System) Server Sensor: 플러그인 센서. CVS 서버의 저장소에서 관리되는 소스 파일의 변경에 대한 정보를 수집한다.
- JUnit Sensor: 플러그인 센서. 유닛 테스트의 결과를 수집한다.
- Source Code Analyzer: 플러그인 센서. 소프트웨어 크기 및 품질과 복잡도를 측정하는 센서이다.
- Eclipse Sensor, Visual C++ Sensor: 플러그인 센서. 수정중인 소스 파일에 대한 정보를 수집한다.

2.2 클라이언트 매니저(Client Manager)

클라이언트 매니저는 여러 개의 센서에서 보낸 데이터에 대해 필요한 프리프로세싱(pre-processing)을 하고 프리프로세싱 된 데이터를 주기적으로 서버에 보내어 데이터베이스에 저장되도록 한다. 또한 캐시 기능으로서 컴퓨터가 네트워크에 연결되어 있지 않은 경우 센서에서 전달된 데이터를 보관하고 있다가 네트워크에 연결된 경우 모아둔 데이터를 서버로 전송한다.

2.3 서버(Server)

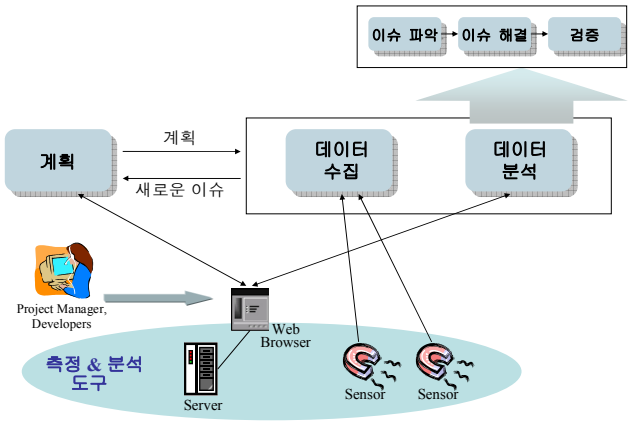
서버는 여러 개의 클라이언트 매니저로부터 전송되는 메트릭 데이터를 데이터베이스에 저장하고 메트릭 데이터의 분석과 결과 제공 및 사용자와 프로젝트 관리 기능을 웹 인터페이스를 통해서 제공한다. 서버는 사용자의 다양한 질의를 받아서 저장된 메트릭에 대한 분석 결과를 웹 페이지를 통해 사용자에게 전달해 준다. 또는 사용자는 분석 방식을 미리 서버에 정의하여 분석 결과를 이메일을 통해서 주기적으로 받아 볼 수 있다.

2.4 데이터베이스(Database)

데이터베이스는 사용자와 프로젝트 관련 데이터 및 수집된 메트릭을 저장한다.

3. 소프트웨어 프로젝트 관련 문제 인식과 문제 해결을 가이드 해주는 방법

본 논문이 제안하는 문제 인식과 문제 해결을 가이드 해주는 방법은 객관적인 문제 파악과 의사 결정을 가능케 하기 위해서 센서로부터 자동으로 수집된 소프트웨어 프로세스와 제품 메트릭에 기반한다는 것이 특징이다. 프로젝트에 참여한 각 개발자의 개발 활동과 관련된 메트릭이 자동으로 수집되어 데이터베이스에 저장되고, 저장된 메트릭을 분석하여 다양한 분석 결과를 개발자 또는 프로젝트 관리자에게 제공함으로써 객관적으로 이슈를 파악할 수 있게 한다. 또한, 파악된 이슈를 해결하기 위해 수행한 활동이 올바른 결과를 가져오는지를 계속적으로 수집되는 메트릭 데이터의 분석으로부터 알 수 있다. 측정 및 분석 프로세스에는 (그림 2)와 같이 계획, 수집, 분석 활동을 포함한다.



(그림 2) 측정 및 분석 프로세스

3.1 측정 및 분석 활동 계획

측정 및 분석 활동 계획에서는 측정의 목적을 파악하고(GQM[11]) 파악된 목적의 추적에 필요한 센서를 결정하고 특정 메트릭에 대한 한계값을 지정한다. 또한 분석 결과를 이메일을 통하여 주기적으로 받기 위하여 필요한 메트릭과 분석 기법을 지정한다. 그 외에 프로젝트 정보 입력, 개발자/관리자 등록을 계획 단계에서 한다. 계획 활동은 프로젝트 초기에 주로 이루어지나 새로운 이슈의 발생을 지속적으로 모니터링 하기 위해 프로젝트 중에도 일어날 수 있다. 필요한 센서를 추가하거나 주기적인 분석 결과를 추가로 등록할 수 있다.

3.2 데이터 수집

메트릭 데이터의 수집은 센서에 의해서 개발자의 개입이 필요 없이 자동으로 수집된다. 소프트웨어 측정 및 분석 활동에서 원하는 결과를 얻기 위해서는 체계적인 메트릭 수집 활동과 신뢰성 있는 데이터의 수집이 가장 중요한 요소임과 동시에 가장 많은 노력을 요구하는 부분이기도 하다. 이러한 어려움을 센서를 통해 해결할 수 있다.

3.3 데이터 분석

분석 단계에서는 수집된 데이터의 분석과 해석을 통해 객관적인 데이터 기반의 이슈 발견과 해결을 하는 것이 주 목적이다. 또한 이슈 해결에 대한 검증 활동도 포함한다. 메트릭의 발생과 동시에 수집과 분석을 할 수 있으므로 이슈의 조기 발견이 가능하게 된다. 개발자나 프로젝트 관리자는 필요 시에 서버에 접근해서 수집된 메트릭 데이터의 분석 결과를 볼 수 있고, 또는 이메일을 통해서 주기적으로 분석 결과를 받을 수 있다. 메트릭 데이터의 분석은 주로 다음과 같은 두 가지 형태가 된다.

- 메트릭 데이터의 흐름 분석: 시간의 경과(예, 시간/날/주/월마다)에 따라서 메트릭 데이터가 어떻게 변했는지를 살펴봄으로써 이슈를 파악할 수 있다. 또한 계획 값이 있는 경우 계획 대비 실제 값의 차이를 살펴봄으로써 이슈를 파악할 수 있다.
- 메트릭 데이터의 한계값 분석: 특정 메트릭에 대해서 한계값을 지정한 후, 실제 수집되는 메트릭 데이터가 이 한계값을 넘어가는 지 아닌지를 살펴봄으로써 이상 현상을 파악할 수 있다. SPC (Statistical Process Control[6])를 이용하여 한계값 분석을 할 수 있다.

흐름 분석과 한계값 분석을 통해서 이슈를 파악한 후 이슈 해결을 시도한다. 이슈 해결에서는 메트릭 데이터의 분석과 해석을 통해 문제의 원인을 파악하여 문제를 해결한다. 이슈 해결에서 올바르게 문제를 해결하였는지는 계속적으로 수집되는 메트릭 데이터를 분석함으로써 알 수 가 있다.

분석은 개발자 개인 레벨에서의 분석과 프로젝트 레벨에서의 분석이 가능하다. 개발자는 자신의 개발 활동과 관련된 메트릭에 대해 분석을 할 수 있고 다른 개발자와 관련된 메트릭에는 접근할 수 없다. 프로젝트 관리자도 개발자 개인의 메트릭에 접근을 할 수 없다. 이는 개발자 개인의 메트릭이 개인의 성과 평가에 쓰일 수 있는 부작용을 막아준다. 프로젝트 관리자와 개발자들은 프로젝트 레벨에서의 분석을 할 수 있다. 예를 들어 모든 개발자들이 발생시킨 결함의 총 개수를 주기적으로(예 날마다) 추적할 수 있다.

4. 관련 연구

4.1 Hackystat and Software Project Telemetry

Hackystat 는 PSP 메트릭(시간, 크기, 결함)을 자동으로 수집하고 수집된 데이터를 분석해 주는 도구[7]이다. 센서라는 프로그램이 개발 도구에 플러그인 형태로 삽입되어 PSP 메트릭을 자동으로 수집한다. 수집된 데이터는 서버를 통해 데이터베이스에 저장되고 개발자는 웹 브라우저를 통해서 수집된 데이터를 분석할 수 있다. 이 도구의 확장된 버전은 PSP 메트릭과 함께 Commit(소스 형상관리 도구에서 commit 이벤트에 대한 메트릭 데이터), Build(소스 코드 빌드와 관련된 정보) 등의 메트릭을 수집하고 프로젝트 팀 레벨에서의 분석 결과를 제공한다. 이 확장된 도구는 수집된 메트릭을 기반으로 프로젝트 관리에서 객관적인 의사 결정을 가능케 하는 새로운 개념의 프로젝트 관리 기법인 Software Project Telemetry[8] 연구에 사용되고 있다. Software Project Telemetry 는 다음과 같은 특징을 가진다.

- 메트릭 데이터는 소프트웨어 개발 환경에서 프로젝트의 상태를 모니터링 하는 툴(센서)에 의해서 자동으로 수집된다.
- 메트릭 데이터는 분석을 위해 타임스탬프 (time-stamp) 정보와 함께 저장된다. Software Project Telemetry 는 시간의 변화에 따른 메트릭의 변화에 중점을 둔다.
- 메트릭 데이터는 계속적으로 수집되고 개발자와 관리자는 수집된 데이터에 즉시 접근할 수 있다.

Software Project Telemetry 에서 제안하고 있는 프로젝트 관리는 문제 발견, 문제 해결을 위한 가설 생성, 프로세스 변경, 가설 검증, 결과 분석의 과정을 포함한다.

4.2 PROM (Pro Metrics)

PROM[9]은 소프트웨어 제품과 프로세스 메트릭을 자동으로 수집하는 데이터 수집 및 분석 도구이다. 수집된 데이터는 PSP 메트릭, 프로시저와 객체 지향 메트릭, 워드 프로세스를 가지고 문서 작업 등의 코딩 작업 이외의 활동을 추적하기 위한 특별 메트릭을 포함한다. PROM 은 다음의 네 가지 컴포넌트로 이루어져 있다.

- PROM Probes: 개발 관련 도구에 플러그인 형태로 부착되어 관련 메트릭을 수집하는 PROM plug-in 과 운영체제의 시스템 호출을 감시하는 PROM Trace를 포함한다.
- PROM Transfer: 수집된 데이터를 플러그인으로부터 받고, 필요한 프리프로세싱(pre-processing)을 한 후, 프리프로세싱된 메트릭 데이터를 PROM Server로 보낸다.
- PROM Server: PROM Database에 데이터를 저장하는 역할을 하며 분석 및 프로젝트와 사용자 관리 등의 기능을 웹 서비스를 통해서 제공한다.
- PROM Database: 사용자와 프로젝트 관련 정보, 수집되는 메트릭 데이터와 분석 결과를 저장한다.

개발자와 프로젝트 관리자는 웹 페이지를 통해서 PROM Server 에 원하는 종류의 분석을 요청할 수 있다. 예를 들어, 코드의 길이, 복잡도, 커플링(coupling)을 나타내는 그래프, 한 프로젝트에 쓰여진 전체 시간 등 원하는 분석을 웹 인터페이스를 통해서 PROM Server 에 요청할 수 있다. 또한 제 3 자(third-parties)가 제작한 분석도구를 이용해서 원하는 분석을 할 수 있게 하는 메커니즘을 제공한다.

센서를 이용한 메트릭의 자동 수집과 서버를 통한 수집된 메트릭의 분석을 제공한다는 점에서 Hackystat 와 PROM 도구는 본 논문에서 제시한 도구와 비슷하다. Software Project Telemetry 는 메트릭 기반의 프로젝트 관리 기법이라는 점에서 본 논문이 제안하는 접근 방법과 유사하다. 본 논문에서는 다음과 같은 점에서 위 연구들의 확장을 제안한다.

- 센서의 확장: Hackystat 도구에서는 수정되는 파일에 대해서만 시간 사용량을 측정한다. 본 논문이 제안하는 도구는 기술 문서 등의 문서 읽기와 파일의 수정 또는 읽기를 포함하지 않는 작업 (예, Internet Explorer의 사용)에 대해서도 시간 사용량을 측정하려고 한다.
- 분석 기능의 확장: Software Project Telemetry는 시간 경과에 따른 메트릭 데이터의 흐름 분석만을 포함한다. 본 논문에서는 SPC의 적용을 통한 이상 현상 파악을 제안한다. 또한 계획 데이터를 이용해 프로젝트가 계획대로 진행되고 있는가를 파악하려고 한다(예, EVMS[10]).

5. 결론 및 향후 연구

본 논문에서는 센서 기반의 메트릭 자동 수집 및 분석 도구와 이 도구를 기반으로 하여 소프트웨어 프로젝트 관련 문제 인식과 문제 해결을 할 수 있게 하는 방법을 제안하였다. 본 논문에서 제안한 도구와 방법을 통해 메트릭 수집에 대한 오버헤드를 없애 지속적이고 체계적인 소프트웨어 개발 프로세스 측정 및 분석 활동을 할 수 있다. 또한 신뢰성 있는 메트릭 데이터의 수집이 가능하여 이 수집된 메트릭의 분석을 통해 객관적으로 이슈를 파악하고 해결 할 수 있다. 또한 위험 요소의 조기 식별과 해결이 가능하여 문제해결에 드는 비용의 절감 효과를 얻을 수 있다.

소프트웨어 제품 및 프로세스 메트릭 측정 및 분석은 프로젝트 관리 이외에도 프로세스 개선에 있어서도 반드시 필요한 부분이다. 수집되는 메트릭의 분석을 통해서 개선할 필요가 있는 프로세스의 요소를 파악할 수 있고, 시도한 프로세스 개선이 실제로 개선의 효과를 가져왔는지를 알 수 있다. 본 연구가 제안하는 센서 기반 메트릭 수집 기법의 측정 및 분석 도구를 확장하여 프로세스 제어 및 프로세스 개선을 할 수 있는 방법에 대한 연구는 향후 가치 있는 연구가 될 것이다.

참고문헌

[1] T. DeMarco. *Controlling Software Projects*, Yourdon Press, 1982.

[2] J. McGarry, D. Card, et al., *Practical Software Measurement*, Addison Wesley, 2002.

[3] M. B. Chrissis, M. Konrad and S. Shrum, *CMMI - Guidelines for Process Integration and Product Improvement*, Addison-Wesley, 2003.

[4] W. S. Humphrey, *PSP^{SM}: A Self-Improvement Process for Software Engineers, Addison-Wesley*, 2005.

[5] El Emam, K. Drouin, J-N. and Melo, W., *SPICE: The Theory and Practice of Software Process Improvement and Capability Determination*, California: IEEE Computer Society. 1998.

[6] W.A. Florac and A.D. Carleton, *Measuring the Software Process: Statistical Process Control for Software Process Improvement*, Addison-Wesley, 1999.

[7] P. M. Johnson, H. B. Kou, J. M. Agustin, C. Chan, C. A. Moore, J. Miglani, S. Zhen, and W. E. Doane. Beyond the personal software process: Metrics collection and analysis for the differently disciplined. In *Proceedings of the 2003 International Conference on Software Engineering, Portland, Oregon*, May 2003.

[8] P. M. Johnson, H. Kou, Michael G. Paulding, Q. Zhang, A. Kagawa, and T. Yamashita. Improving software development management through software project telemetry. *IEEE Software*, August 2005.

[9] Sillitti A., Janes A., Succi G., Vernazza T., “Collecting, Integrating and Analyzing Software Metrics and Personal Software Process Data”, *EUROMICRO 2003*, Belek-Antalya, Turkey, 1 - 6 September 2003.

[10] Lett, Steve, “Earned Value Management for Self Directed Software Teams,” Software Engineering Process Group, Lockheed Martin.

[11] Fenton NE and Pfleeger SL, *Software Metrics: A Rigorous and Practical Approach (2nd Edition)*, International Thomson Computer Press, 1996.