Improving Software Development Management through Software Project Telemetry

Philip M. Johnson   Hongbing Kou   Michael Paulding
Qin Zhang   Aaron Kagawa   Takuya Yamashita
Collaborative Software Development Laboratory
Department of Information and Computer Sciences
University of Hawai'i
Honolulu, HI 96822
johnson@hawaii.edu

Abstract

Software project telemetry is a new approach to software project management in which sensors are attached to development environment tools to unobtrusively monitor the process and products of development. This sensor data is abstracted into high-level perspectives on development trends called Telemetry Reports, which provide project members with insights useful for local, in-process decision making. This paper presents the essential characteristics of software project telemetry, contrasts it to other approaches such as predictive models based upon historical software project data, describes a reference framework implementation of software project telemetry called Hackystat, and presents our lessons learned so far.

1. Introduction

It is conventional wisdom in the software engineering research community that metrics can improve the effectiveness of project management. Proponents of software metrics quote theorists and practitioners from Galileo's "What is not measurable, make measurable" [3] to DeMarco's "You can neither predict nor control what you cannot measure" [1]. Software metrics range from internal product attributes, such as size, complexity, and modularity, to external process attributes, such as effort, productivity, testing quality, and reliability [2].

References

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

[2] N. Fenton and S. L. Pfleeger. *Software Metrics: A Rigorous and Practical Approach*. Thomson Computer Press, 1997.

[3] L. Finkelstein. What is not measureable, make measurable. *Measurement and Control*, 1982.

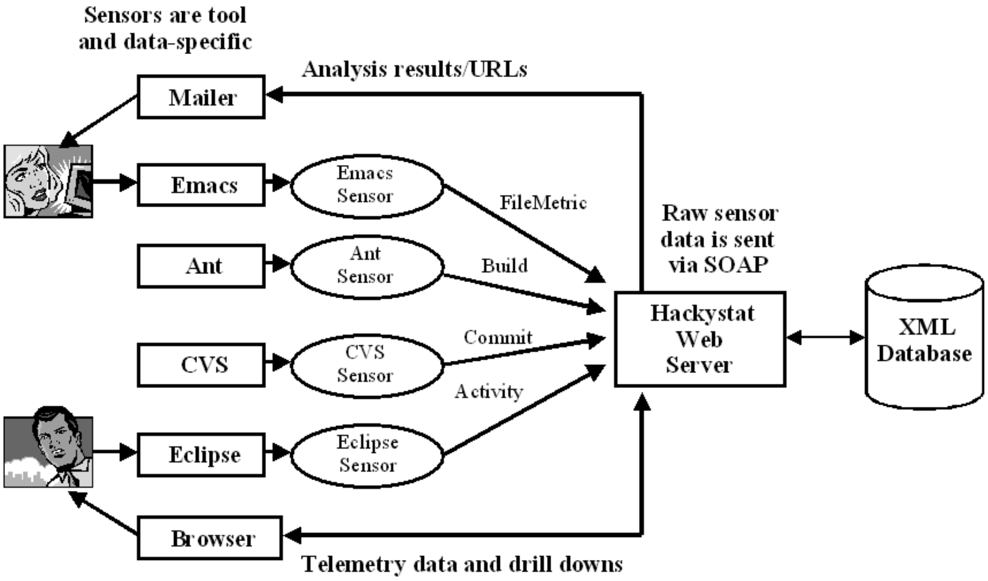

Figure 1. The basic architecture of Hackystat. Sensors are attached to tools directly invoked by developers (such as Eclipse or Emacs) as well as to tools implicitly manipulated by developers (such as CVS or an automated build process using Ant).