STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LV**, Number 2, 2010

SMART SENSORS FOR SOFTWARE TELEMETRY SYSTEMS

ADRIAN ROMAN

Abstract. Software telemetry is a software project management approach that uses sensors attached to development tools to monitor and control the development process and the resulted products. Software telemetry proposes solutions for metrics collection cost problem and metrics decision-making problem, but it also comes with several downsides. This article identifies the problems of software telemetry and proposes the concept of smart sensors as possible solution for improving the approach.

1. Introduction

1.1. **Software measurements.** Software development is known as a slow and expensive process, often resulting in low quality products with serious reliability, usability, and performance problems. For decades, software development researchers have been attempting to control and improve software development processes and increase the quality of the final products guided by DeMarco's saying[1], "*You can neither predict nor control what you cannot measure*".

Measurement is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules [2]. Software measurements refers to the characteristics of a software product or the software process and make the project management more effective. However, practitioners face various barriers in applying metrics. Due to the high cost associated metrics collection and difficulties in using them in the decision-making process, measurements are not widely used within the software organizations.

Software telemetry, a new approach in the software measurement field, is a step towards solving both the metrics collection cost problem and metrics

Received by the editors: 2010/4/25.

2010 *Mathematics Subject Classification.* 68N30, 68U35.

1998 *CR Categories and Descriptors.* K.6.3 [**Management of Computing and Information Systems**]: Software Management – *Software development*; K.6.3 [**Management of Computing and Information Systems**]: Software Management – *Software process*; D.2.8 [**Software**]: Metrics – *Process metrics*.

Key words and phrases. Software telemetry, software project telemetry, software sensor, smart sensor.

decision-making problem. Software telemetry also comes with several downsides, related to the way sensors are implemented and also related to the set of metrics collected.

This article identifies the existing problems of the software telemetry and introduces the smart sensors as possible solution.

1.2. **Software telemetry.** Software project telemetry is an approach to software project management that uses sensors to collect software metrics automatically and unobtrusively. The following important properties define software telemetry as a style of software metrics definition, collection, and analysis[3]:

- *Automatic collection of the data by sensors attached to tools which measure various characteristics of the project environment on constant basis.*
- *The data is a stream of events. Each event is time-stamped which is significant for analysis.* Data analysis in software project telemetry is focused on the changes over time of various parameters.
- *The data is accessible to both developers and the managers.* Project members have to be able to transparently access the collected data.
- *Telemetry analyses exhibit graceful degradation.* The analyses should provide value even if complete data over the entire project's lifespan is not available.
- *Analysis of data includes monitoring and control of processes as well as short-term prediction.* Telemetry analysis represents the project state at a given time and how it changes at various timescales.

Software telemetry is used in projects where the software developers work at a location which is inaccessible for manual metrics collection activities. This approach is in contrast with the software metrics data which is required to be collected by the developer. Furthermore, the software telemetry data has its focus on the changes over time in the measurements of various processes and product during development. In addition, the access to telemetry data is not restricted to software quality improvement groups, all developers and managers of a project can view the data in order to take a decision according to the future requirement of the project. Also, the telemetry analysis shows the current state of the project and the changes that occurs in it over time, the scale of which can be decided. The parallel display of more than one project state values and the pattern of their change allows for opportunistic analysis, which determine how different variable co-vary with each other.

Software telemetry addresses the *metrics collection cost problem* by automated data collection - sensors replace the manual metrics collection activities, thus, reducing operational costs. It addresses the *metrics decision-making*

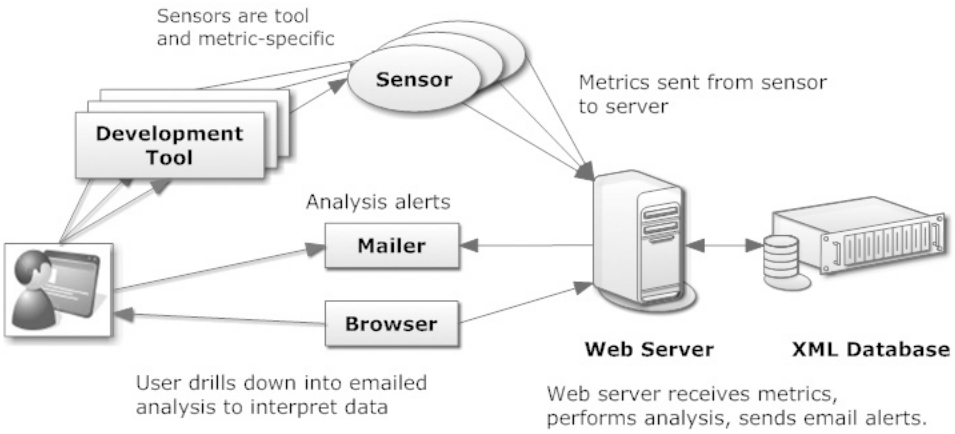

FIGURE 1. The architecture of a software telemetry system[4]

problem through high-level visual perspectives on software development process. Unlike traditional metrics approaches which are primarily based on historical project databases and focused on model-based project comparison, software project telemetry emphasizes project dynamics and in-process control.

1.3. **Software telemetry system architecture.** A software project telemetry system consists of three components: the sensor, the collector (server) application and the data visualization interface. Figure 1 illustrates the architecture of a software telemetry system[4].

Sensors are an essential part of any telemetry system. In software project telemetry systems data is collected automatically by tools / sensors that regularly measure various characteristics of the development environment. Data consists of a stream of time stamped events and developers and managers can immediately access the data.

Sensors are attached to development environments and can be implemented as plug-ins (in certain cases called *add-ins* or *add-ons*). Plug-ins are optional components which can be used to enable the dynamic construction of flexible and complex systems, passing as much of the configuration management effort as possible to the system rather than the user, allowing graceful upgrading of systems over time without stopping and restarting.

The basic function of a sensor is to collect data from the development environment and send it to a centralized collector application. In order to obtain usable and valuable data, more requirements have to be fulfilled by a sensor[6]:

- *It has to be completely integrated to the development environment.* The sensor should not interfere with the development process.
- *It has to be able to send data a centralized warehouse (server application).* A connection between the development environment and the server has to be established and the sensor has to be able to collect the data and send them to the database.
- *It has to be able to send data automatically.* The process of data collection and sending has to be automatic, with no human intervention or effort.
- *It has to be able to timestamp events.* As the data analyses are based on evolution over time, the data has to be time stamped.
- *It has to gather significant project management data (e.g. count the code lines for the current open projects).* The sensor have to be able to collect data useful for analysis, not just any data.
- *A setup has to be provided (easy deployment).* The deployment of a sensor has to be easy to perform.

Some examples of sensors used in software telemetry systems are listed below:

- *A plug-in for an IDE (integrated development environment) such as Visual Studio, and Eclipse.* It can record individual developer activities automatically, such as code editing effort, compilation attempts, and results, etc.
- *A plug-in for a version control system, such as CVS or SVN.* It can monitor code check-in and check-out activities, and compute or show the differences between revisions.
- *A plug-in for a bug tracking or issue management system, such as Bugzilla, and Jira.* Whenever an issue is reported or its status is updated, the sensor can detect such activities and record the relevant information.

1.4. **Downsides of software telemetry.** As with any other approach, the use of software telemetry has several disadvantages attached to it as well. For example, it is very much possible to misinterpret or misuse the software project telemetry data. Furthermore, the adoption of software telemetry approach for decision making and measurement purposes emphasize an increased use of tools to manage process and products which can incur additional cost in the project[3].

The main downsides of software telemetry are sum up below:

- **Sensors are tool and metric-specific. A sensor must be developed for each type of tool to be monitored.** Although, this is one time cost, every time a new development tool has to be monitored, a sensor has to be developed. Moreover, every change in the metrics required by the analysis component or in the development tool architecture results in changes to the sensor level.

- **Some metrics are not suitable for automated collection.** For example, the software development effort. It is almost impossible to construct a sensor that knows how much effort a developer has contributed to a project. The total effort represents not only the development, programming time, but also the analysis, design, thinking effort that cannot be monitored and collected by sensors attached to the development tools. It is still an open research question whether all important metrics can be captured by sensors or not.

The introduction of smart sensors addresses these problems and proposes solutions for further improvements of software telemetry.

2. Smart sensors and proposed architecture

In order to overcome the downsides previously mentioned, an extension of the existing architecture is possible. Sensors with extended properties can be designed and implemented as part of a telemetry system. A so-called smart sensor (could also be called extended sensor) should have the following properties added to the ones mentioned in section 1.2 of this article:

- *It is machine-specific sensor instead of tool-specific sensor.* This means that a smart sensor is able to monitor several applications in the same time and it is independent of the development tools. Thus, sensors are designed and implemented for every type of machine instead of every type of development tool.
- *It is able to collect an extended range of data. Not metric-specific.* The sensor must be capable of collecting a large number of metrics by design.
- *It is able to receive messages from the centralized application.* The smart sensor should be able to receive and interpret messages from the centralized application. The messages should indicate the metrics to be collected by the sensors.
- *It is able to collect and send only the required data.* In order to reduce the data overload, only required data should be monitored and sent to the centralized application.
- *It allows user interaction. The sensor should have a user interface that allows the user interaction.* The interface should allow the input of metrics that cannot be collected automatically and should not interfere with the sensor functioning process.

2.1. **Proposed architecture for smart sensor integration.**

The architecture of software telemetry system [5] described in figure 1 easily integrates such a smart/extended sensor (figure 2). The main components of the system remain the same, only that data flow is a little bit different. Centralized application is responsible for both data collection and sending instructions and

messages to sensors. Management interface is used to sustain the decision process. Project managers are able to tell exactly what metrics they need to be collected for a certain period of time. Smart sensors have all the properties of

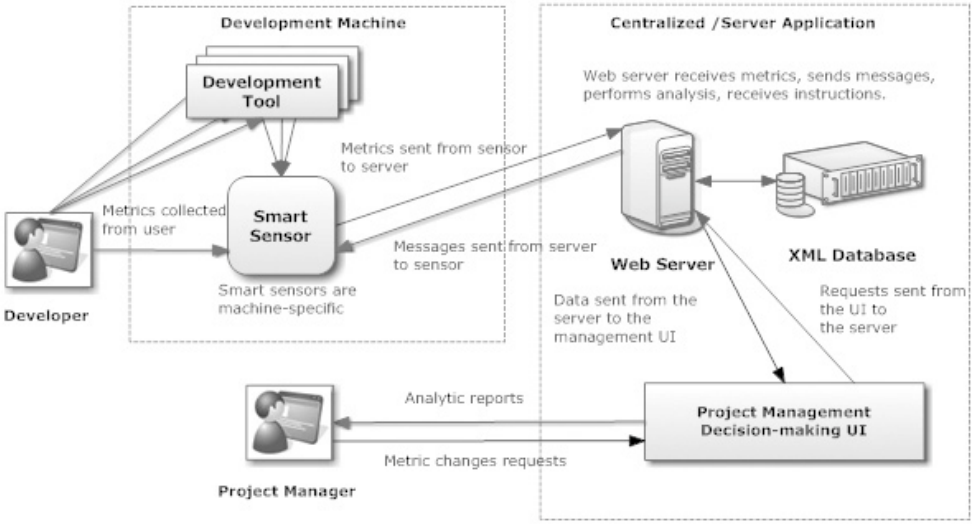

FIGURE 2. Smart sensors in a software project telemetry system

regular software sensors plus several more that are meant to solve both problems identified in Section 1.3. Table 1 shows how each problem is addressed by the smart sensor solution.

Problem	Solution
A sensor must be developed for each type of tool to be monitored	Smart sensors are machine-level. There is no need to develop a sensor for each tool. Smart sensors are able to collect an extended range of data. Not metric-specific. Smart sensors are able to receive messages and change the set of collected metrics dynamically.
Some metrics are not suitable for automated collection	Smart sensors allow user interaction. Thus, metrics that are not suitable for automatic collection can be collected through the user interface and sent to the centralized applications in the same way as data collected automatically.

2.2. **Smart sensor implementation considerations.** Creating software sensors is not easy. The developers need to know a lot about the development environment that the sensor is attached to. Plus, development environments have bugs in their plugin APIs, they are due to frequent changes and most of the time are not well documented.

Smart sensors try to solve these problems by implementing a machine-level application that is able to capture all the necessary information in a generic way, independent from the development environment. Such a sensor should be able to collect metrics like the time spent on a certain application, file or project, the idle time, the number of code lines written for different configuration and development environments. This is the biggest challenge of this approach and further study is needed as it involves high-level programming skills.

Another important feature of a smart sensor is the capability of collecting only specified metrics. The sensors receive messages from the server and collects only certain metrics. A telemetry language for the communication between sensors and the centralized application has to be designed.

The central idea of sensor-based metrics collection is that sensors are designed to collect metrics automatically and unobtrusively. Once they are installed, they work silently in the background. But smart sensor also allow user interaction. Metrics that cannot be collected automatically, such as the time spent for brainstorming or team meetings, can be introduced in the software telemetry system via a thin application that connects the user with the sensor. The data then is forwarded by the sensor to the server application.

Offline storage needs to be also considered for the situations when the development stations fail to connect to the centralized server. When the communication with the server is not possible the metrics are stored locally using XML files, a small database or open source solutions like Google Gear.

3. Conclusions and future work

The use of software project telemetry supports project management decision making. Furthermore, the automated collection of data adds significant value to software telemetry as it makes all metrics more comparable and current. However, there is always a chance that the data obtained through sensors of software telemetry can be misinterpreted or misused and it also increases the dependency of the project team on tools for managing process and products. Hence it can be said that although software telemetry approach does not provide a silver bullet to solve all problems that are associated with metrics-based project management and decision making, however, it does address the inherent problems found in traditional measurement and provides for a new approach to more local, in-process decision making.

Software telemetry presents two main downsides: (1) **sensors are tool and metric-specific** and (2) **some metrics are not suitable for automated collection**. A sensor has to be developed for every type of development tool and every change in the required set of metrics results in changes to the sensor level. Also there are metrics that cannot be collected automatically and require human input.

This article introduces the concept of smart sensors that can be used in software telemetry systems to overcome the above mentioned downsides. A smart sensor is metric-specific, is able to collect a large range of data and allows user input. Thus, sensors are implemented for every type of machine instead of every type of development tool and user can input metrics that are collected, transported and analysed by the telemetry system. Further research is need on the way smart sensors can be implemented to achieve the proposed properties, on the communication protocols to be used and on the architecture of software telemetry systems using this approach.

References

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

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

[3] P. M. Johnson, H. Kou, M. Paulding, Q. Zhang, A. Kagawa, T. Yamashita, *Improving software development management through software project telemetry*, Software, IEEE, 22(4), 2005, pp. 76-85.

[4] P. M. Johnson, H. Kou, J. Agustin, C. Chan, C. Moore, J. Miglani, S. Zhen, W.E.J. Doane, *Beyond the Personal Software Process: Metrics collection and analysis for the differently disciplined*, Proceedings of the 25th International Conference on Software Engineering, IEEE Computer Society, 2003, pp. 641 - 646.

[5] A. Roman, *Proposed Architecture for Software Telemetry Systems*, Proceedings of the National Symposium ZAC 2008, ISBN: 978-973-610-730-6, 2008, pp.31-37.

[6] A. Roman, *Towards Building Software Project Telemetry Tools*, Proceedings of the International Conference "European Integration Between Tradition and Modernity", ISSN: 1844-2048, 2008, pp.731-734.

"Petru Maior" University of Tirgu-Mures
E-mail address: aroman@science.upm.ro