A New Architecture and Implementation Strategy for Non-Invasive Software Measurement Systems

Anton Bykov
Innopolis University
Innopolis, Russian Federation
a.bykov@innopolis.ru

Vladimir Ivanov
Innopolis University
Innopolis, Russian Federation
v.ivanov@innopolis.ru

Alan Rogers
Innopolis University
Innopolis, Russian Federation
a.rogers@innopolis.ru

Alexandr Shunevich
Innopolis University
Innopolis, Russian Federation
shunevich.aj@gmail.com

Alberto Sillitti
Innopolis University
Innopolis, Russian Federation
a.sillitti@innopolis.ru

Giancarlo Succi
Innopolis University
Innopolis, Russian Federation
g.succi@innopolis.ru

Alexander Tormasov
Innopolis University
Innopolis, Russian Federation
a.tormasov@innopolis.ru

Jooyong Yi
Innopolis University
Innopolis, Russian Federation
j.yi@innopolis.ru

Albert Zabirov
Innopolis University
Innopolis, Russian Federation
zabirov.albert@gmail.com

Denis Zaplatnikov
Innopolis University
Innopolis, Russian Federation
dzaplatnikov@mail.ru

ABSTRACT

Despite that non-invasive software measurement tools have proven their usefulness in software production, their adoption in software industry is still limited. Reasons for the limited distributions have been studied and analysed recently. In this paper, we propose a new architecture for non-invasive software measurement systems that address the problems of the existing systems. The outcome of our early experimentation is quite promising and gives us the desired additional confidence on its successful distribution.

CCS CONCEPTS

• **Software and its engineering → Software development process management**; *Client-server architectures*;

KEYWORDS

software measurement, non-invasive sensors, software architecture

ACM Reference format:
Anton Bykov, Vladimir Ivanov, Alan Rogers, Alexandr Shunevich, Alberto Sillitti, Giancarlo Succi, Alexander Tormasov, Jooyong Yi, Albert Zabirov, and Denis Zaplatnikov. 2018. A New Architecture and Implementation Strategy for Non-Invasive Software Measurement Systems. In *Proceedings of ACM SAC Conference, Pau,France, April 9-13, 2018 (SAC'18),* 8 pages.
https://doi.org/10.1145/3167132.3167327

1 INTRODUCTION

Measurement of software processes and products is a well-known way to increasing quality, control and predictability of resulting software [35]. Collecting process and product metrics facilitates reconstructing of a development process and may produce insights on how to improve the productivity of software development and its quality.

However, collecting metrics is also a difficult task [21, 25]. In particular, collecting metrics from the developers successfully depends, by and large, on how much support the developers provide for metrics collection, as witnessed in [5]. Developers would not welcome being disturbed for the sake of metrics collection. Thus, a *non-invasive* collection of software metrics—where metrics are collected automatically, without requiring the personal involvement of the developers—has been affirmed as one of the most promising approaches for metrics collection [14, 26, 36]. By using a non-invasive software measurement system, data about software products and software development processes can be collected from developers' machines, smartphones, smart things, product repositories, and task/defect tracking tools, without disturbing the developers.

Adoption of non-invasive metrics collection systems in software industry is still limited, despite its promising potentials. The case study of Coman et al. [5] sheds light on why this is the case. According to them, the existing non-invasive metrics collection systems need the following for wider adoption: increasing the level of data privacy, letting the developers see/modify the information stored about them, allowing the developers to turn off the system partially or totally when needed, providing an effective user interface to access the data (e.g., a hierarchical view of the data), providing the ability to integrate data from a variety of other systems, and making the system fault tolerant.

We have developed a non-invasive software measurement system that addresses the aforementioned functionalities missing in the existing systems. In this paper, we describe the novel architecture and implementation strategy of our measurement system. The main contribution of the work is focused on the analysis and justification of architectural decisions behind development of a system for non-invasive software measurement.

This paper is organized as follows. Section 2 describes the current state of the art. Section 3 outlines the new proposed architecture. Section 4 presents the implementation details. Section 5 summarises the data from the early implementation of the system. Section 6 discusses the early results of the adoption of the system. Section 7 draws some conclusions and outlines the main directions of future research.

2 SURVEY OF THE STATE OF THE ART

Understanding and controlling software development process is, despite its importance, quite difficult due to the high complexity of software and the high degree of uncertainty software developers experience. To understand and control software process better, measuring software metrics – a collective term used to describe the wide range of activities concerned with measurement in software engineering [10] – can be a good starting point.

The history of software metrics collection can be divided into two generations [19]. The first generation applies the Personal Software Process (PSP) – a self-improvement process that helps developers to control, manage, and improve the way they work [12]. PSP can also be called an "invasive" method of metrics collection since it requires the direct involvement of participants in the data collection process. Users of the PSP create and print forms in which they log their effort, size and defect information. One obvious downside of this invasive approach is the high overhead cost it entails. The developers should often switch between development tasks and metrics collection tasks, which imposes a high cognitive burden to the developers [24].

To reduce metrics collection cost, the more recent second-generation approach collects metrics from the users in a "non-invasive" way where software metrics are collected automatically, without requiring the personal involvement of the users in the data collection process. Table 1, which is an adapted version of a table that appeared in [19], illustrates distinguishing characteristics of invasive and non-invasive approaches. It becomes clear that the non-invasive approach reduces costs of collection and analysis process as well as context switching problem (when the developer should switch from working process to filling PSP forms).

Ways of implementing non-invasive measurement are discussed in [15, 29]. Non-invasive collecting systems should focus on the following aspects to satisfy characteristics shown above [31]:

- automatic collection of product metrics;
- support of the tools that are used by the developers;
- support of the programming language used by the developers;
- automatic installation and update of the tools for data collection.

Table 1: Characteristics of invasive and non-invasive measurement

Characteristic	Invasive approach	Non-invasive approach
Collection overhead	High	None
Analysis overhead	High	None
Context switching	Yes	No
Metrics changes	Simple	Tool dependent
Adoption barriers	Overhead, Context-switching	Privacy, Sensor availability

To accomplish these objectives, measurements should be provided through measurement probes or sensors. These probes are put into the software development process, then report events to a central repository where the data can be analyzed and shown. There are two common ways to extract data:

- batch mode – the data are extracted on a regular basis;
- background mode – the data are extracted continuously.

Batch mode is useful when it is not necessary to track and collect an ongoing process or if the collecting process requires a lot of computer resources and will be costly in terms of performance. Background mode works in the opposite way and collects data as soon as it become available.

And two ways to submit data to central repository:

- online – the collected data are immediately submitted to the server;
- cached – the collected data are saved locally and then submitted later.

Cached approach is useful for devices that do not have a constant connection to the network, so it will be useful to store data locally and send it when a connection become available. If the bandwidth is low – a cached approach can collect, compress and send data later. It is also possible to allow manual input of data. In this case it is just needed to implement the mechanism that will collect manual input and send it to the server in the same format as non-invasive measurements.

The description of common approaches is provided in [15]. But concrete implementation of the measurement process depends on the customer's requirements. A measurement system for a small team will have different infrastructure than a system for a big company. Even if team sizes are nearly the same they may have different goals and make measurements in different ways. For now, PRO Metrics (PROM) and Hackystat are the most widely known collection systems.

PRO Metrics, described in [27, 28, 31] is a distributed architecture for collecting different kinds of software data: software metrics and PSP data. PROM is based on Package-Oriented Programming development technique.

PROM includes four main components:

- PROM plug-ins for IDE that track application-generated data, collect and send all these data to PROM Transfer with timestamp and user authentication features. These authentication feature are different from traditional ones, so they allow multi-user logins to 'Agile' practices such as pair programming [8, 16].
- PROM Trace plug-in allows to track interesting operating system calls and user interaction with the system. It can track the name of the current window in focus, browser tab, etc.
- PROM Transfer gets data from all plug-ins and makes preprocessing of collected data to remove redundancy. It sends processed data to PROM Server and stores it in database. PROM transfer can also work offline – it collects data in local storage and sends it when the PROM Server is available.
- PROM Server provides an interface to the PROM Database through Web Services. PROM Database stores all information about users and projects.

For cross-platform support, Update Manager is written in Java and launched through web interface. The main purposes of PROM Update Manager are:

- support first installation of the PROM client-side components (including the data transfer tool);
- automatically update installed components.

Johnson et al. in [18–20] described the Hackystat system. This system automatically collects development metrics from sensors (attached to development tools) and sends them to the server where this data can be analyzed.

In the first version of Hackystat, its sensors were able to collect:

- activity data (e.g., which file is under modification of developer at 30 seconds interval);
- size data (e.g., lines of code);
- defect data (e.g., number of pass/fail status of unit tests).

Developers should install one or more sensors to begin using Hackystat and then register with its server. After that they can start working and metrics will be sent to the server:

- automatically in some intervals (if connection to the network is available);
- or cached locally for later sending.

Hackystat also has one interesting mechanism – the ability to define alerts, which are periodically analyzed based on developer's data. If some sort of threshold value is exceeded the system will send an email (obtained from registration) to the developer that will alert her and will contain a link to the more complete data observations.

A system that will have the following properties will combine characteristics of profitable non-invasive metrics collection written above and use best practices of existing systems:

- has client-server architecture;
- has client-side application as simple as possible;
- uses sensor-based approach (a sensor should be integrated into OS environment);
- sends metrics collection data to the server in JSON format since it is very extensible for new types of data.

3 OUR NEW ARCHITECTURE FOR NON-INVASIVE MEASUREMENT SYSTEMS

3.1 Main Novelties of Our Approach

A critical prerequisite for the successful use of a non-invasive measurement system is the support from the developers in the company. Without their cooperation, metrics cannot be collected, despite the availability of a non-invasiveness of a metrics collection system. However, it is a common concern that the developers often fear that their data might be misused by the "Big Brother" and refuse to use a metrics collection system. We address this concern by:

(1) Ensuring data privacy — every developer has his own account which can be accessible only by him, so his data is safe from unauthorized access;
(2) Giving full control over the developer's own data – the developer can choose which data she would like to send to the server for further analysis [9, 22]. Such selectivity is achieved by applying filters on the collected data (by time interval, keywords or name of the activity) or by manually removing unwanted records.

Our solution, while seemingly simple, is solidly based on the results of the case study conducted by Coman et al. [5]. To the best of our knowledge, we for the first time report in the literature the use of the aforementioned approach in a non-invasive software measurement system.

Another important issue we consider is that our software measurement system should collect metrics from various sources (e.g., web browsers and editors). Our system architecture should allow to collect different metrics from different sources. To meet this need, we split the client side application into the following two parts: collector and manger. First, the collector part collects metrics from its source application, and store them in a local database, which is shared between all collector instances running in the client system. Meanwhile, the manager part transfers the data in the local database to the server side. It also enables the developers to review the data collected from collector instances, before the developers allow all or part of the data to be transferred to the server.

Furthermore, the system should be self monitoring and self healing [5]. If the operating system has crashed and restarted, the measurement system will be restarted without any user interaction with help of the auto-start feature.

Moreover, the client-side system can be used not only with our server side analytic, but can be integrated with any external server which will implement processing of the corresponding format. This will give the ability to include new ways of collecting data into companies' existing systems (if they have any). The systems can be extended and improved since they are open-source. Such extensions will help companies to spend less effort to develop their systems to collect domain specific metrics.

3.2 Server Side

The main problem the server side is supposed to solve was the absence of an architecture for flexible metrics storage. The system should have an ability to store all possible data structures and an ability to handle runtime errors or input errors without data loss.

Such an effect was achieved through architectural solutions. All known data structures were designed in order to meet 3rd normal form, and unanticipated ones can be stored as additional entities with properties “name", “type" and “value".

3.3 Client Side

One of the primary problems which we solved with the client side system is to collect the data in a non-invasive way, while also giving the programmer the ability to control the process. The client side was implemented as a service or status bar application so that a user need not be concerned while the system collects the data. The user can still control the collection process and pause or even stop the collection at any time she wants.

The second problem which had not been solved previously is that the user should be able to filter or remove data after it has been collected. This system provides the ability to filter records; filtering means that records are retained in local storage but removed from the set that will be sent to the server based on:

(1) period of time;
(2) application name;
(3) keywords.

Finally, the client application also allows the user to permanently remove any record from local storage which she doesn't want to send.

The client side system is split into two main parts:

(1) The collector application which is responsible for collecting user activities.
(2) The manager application which is responsible for authorization, managing collected data (filtering and deleting) and sending the data to the server.

Implementing such an approach gives the system modifiability and separates passive data collection, which requires no user interaction, from the management of collected data, which the user should be able to review (if she wants to modify some records) before sending the data to the server.

It was decided to use a shared database for Collector and Manager applications, since there is large overlap between the data they use. Both applications operate within the same domain models and use the same data entities. This approach also reduces space overhead because everything is stored in one place, and avoids the need to spend extra time copying duplicated data from Collector to Manager.

The client side system gives the developer full control over collecting the data and allows her to choose what records she wants to send to the external side. The system can be easily extended to include additional types of data by simply implementing a collection mechanism for the new data.

4 IMPLEMENTATION DETAILS

4.1 Server Side

Back-end data structure can be described with the entity-relationship diagram (see Figure 1)

Users. Users is a model for collecting information about users: name, online contact information, join date, etc. This model is

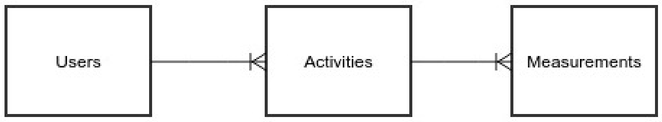

Figure 1: Back-end database schema

needed to group activities by user and to look through personal metrics of each user.

Activities. Activities is a model which stores information about users' activity: name of the activity and an extra field “comments" if the activity has some extra information.

Users-Activities relationship. One user may be related to many activities, but activities have only one owner. So, Users-Activities has the one-to-many relationship. From database point of view, table “activities" will store the foreign key to “users" table.

Measurements. Measurements is a model with the largest amount of tuples. It contains serialized data about the measurement of an activity. In this model, all the information about activities can be stored. For example, it can contain information about the duration of activity, MAC address, and IP-address.

Activities-Measurements relationship. One activity may have many different measures, but measures relate to only one activity. So, Activities-Measurements has the one-to-many relationship. From database point of view, table “measurements" will store the foreign key to “activities" table.

4.2 Client Side for Mac

It was decided to write the system in Swift programming language because it gives developers an ability to write code that is fast, safe, maintainable, and compatible with Objective-C.

In order to focus on business logic and create an abstract layer for the database models, the Core Data framework [1] was used. From the types of data storage provided by the framework, SQLite was chosen, because this is a lightweight solution that does not require the installation of an additional environment and uses the full power of SQL and relational databases. Thus, the collected data can be used outside the system and analyzed using third-party tools using standard SQL queries. The database stores the information about user activities which includes application name, bundle name, bundle path, timestamp when application became the frontmost, timestamp when focus switched to another application, duration when application was the frontmost one, and browser tab name and url (only collected for Safari and Chrome). It also stores data about the user and her computer such as OS version, host name, user login, IP-address, and MAC-address.

The Metrics Collector application is implemented as a Mac Status Bar Application - an application that can be seen on the right top of the status bar in macOS. This kind of application is an excellent choice for tasks that must be performed in the background (collection of activities), and with which the user should be able to interact. The user can pause or completely stop collecting metrics by clicking on the Pause or Stop button, respectively. It also displays information about the current session and the time spent in the current

[1]https://developer.apple.com/reference/coredata

process. Moreover, the application supports the startup function, so the user does not need to think about starting the Collector after rebooting the system, this will be done automatically.

Following the Observer pattern, Apple created an NSNotificationCenter object, a mechanism that provides the ability to send and receive broadcast messages inside and outside the application. One of such messages is NSWorkspaceDidActivateApplication, which "is sent when the Finder is about to start the application." [2] This is the main mechanism which is used to track switching between applications. Without it, the Collector would be forced to poll the state of the system every N seconds, which, firstly, would load the user's computer, and, secondly, would require the application logic to become more complex.

A special case of activities is the browsers - for them it is needed to collect additional metrics: the name of the tab and its url. There are no standard tools for obtaining this information, so it was decided to use AppleScript, a scripting language which allows to control applications that support it. Unfortunately, not all browsers support AppleScript, but Safari and Chrome (the most popular on the macOS platform) support its use. Thus, using a small script, it is possible to get the required browser metrics using the following algorithm:

(1) When a new application becomes active, Collector checks if it is Safari or Chrome.
(2) If it's Safari or Chrome, a background task is created that pulls the tab name and its url from the corresponding browser.
(3) Every 5 seconds, this background task checks if the active application remains with the browser that it was created at run-time, executes scripts, and writes the collected metrics to the database.
(4) If the new activity is not the browser for which the background thread was created, then the thread is terminated.

The five-second polling interval is based on the assumption that if the user does not spend more than 5 seconds in one tab, then this activity is not important and it is permissible to lose it. This interval allows not to load the system with frequent requests to the browser state.

To manage the collected data, the desktop application Metrics Manager was developed. Before the user can manage the collected data, he must be authorized. To do this, he enters his login and password, the application sends a POST request to the server with the appropriate user data. In case of a successful response from the server, the application saves the authorization token, which will be required later to send the collected activities, and shows the user the main application screen. If the authorization was unsuccessful, the user receives a corresponding message asking him to enter the data again.

To implement the update mechanism, Sparkle framework was used [3] - an open-source project that is already used by Evernote, SourceTree, TeamViewer and many others. This framework delivers updates using appcasting - the practice of using Rich Site Summary (RSS) to provide information about the update. Sparkle is integrated into both the Collector and Manager applications, so remote clients will always be up to date. Applications check for updates every time they run or every hour if an Internet connection is available.

[2]https://developer.apple.com/reference/foundation/nsnotification.name
[3]https://sparkle-project.org

4.3 Client Side for Windows

For implementation technology the .NET Framework and C# programming language were chosen. For convenience of implementation of interaction with a database, it was decided to use an object-relational mapping (ORM) tool, and LINQ To SQL was chosen as the most suitable because it is a lightweight and straightforward solution intended for client side applications.

The whole Windows Agent system consists of 2 Windows Forms applications:

(1) Metrics Collector Application - to collect information about users' activities.
(2) Metrics Sender Application - to manage information about users' activities (presentation on the client and transmission to the server) and to provide an update mechanism for the whole system.

The Collector gathers data in response to events (left click and active window change) and at intervals using a timer running in the background. When triggered, the Collector gathers data about:

- Window instances (name, ID, executable path, text);
- System state (user name, IP address in local network, MAC address of WiFi module);
- url from Google Chrome browser;

Data captured by the Collector is written to the local database. The events to be collected should be chosen before collection starts. This approach was chosen for logical and implementation simplicity: first the events to be collected are chosen, and only then does the Metrics Collector Application begin collection. To perform any changes the collection process is stopped, then re-initiated to begin collecting the newly specified data. Data is collected as snapshots of the current system state, and a snapshot is represented by a Registry class.

On request by the client, the data from the Collector snapshots is transformed into the format of activities, which record the duration for which a particular window or page was active. The Metrics Sender Application provides authorization with and transmission to the Server application as JSON strings.

Storing mechanism is represented by the *Writer class*. The aims of the class are:

- Working with the storage (with the database using *MetricsDataContext* class, which provides the database queries interface), in particular:
  - creating the database;
  - saving data into the database.
- Accumulating the snapshots for future saving.
- Performing the saving action iteratively after a given interval (provided with *Guard* class)

Metrics Processing library represents all the logic for the transformation of snapshots into the format of activities. The aim of it is to take snapshots from storage, process them and return a list of activities, which is represented by *ActivitiesList* class. Processing is performed on a request from a client part, which uses that mechanism, and it is an indivisible operation. Notably, the client provides the following filtering parameters before processing:

- Name filter - a list of strings; if a window title contains (as a substring) some string from the list - that entry will be filtered out.
- A parameter which defines, if NULL titles should be filtered out or not.
- "From" and "until" time; only registries (snapshots) within the borders will be considered, all the registries (snapshots) beyond will be filtered out.

The steps of the processing algorithm:

(1) Obtain all the non-processed registries (snapshots) from the database as a list of registries (RL).
(2) Perform filtering using the given parameters.
(3) While RL is not empty do:
   (a) Detect the first activity - all the consecutive registries (snapshots) beginning from the first, which have the same window title (and browser tab name, in the case of a browser).
   (b) Extract the detected activity - replace all the registries (snapshots) of the detected activity from RL to another list (L).
   (c) Transform registries (snapshots) from L into a single activity (which contains all the information described in functional requirements).
   (d) Add the activity obtained in the previous step to a list of activities (AL).
   (e) Store IDs of all the registries (snapshots) from RL into a special container in AL - *RegistriesIds*. RegistriesIds - IDs of registries which were used the activities. They can be used for deletion of registries or updating processed status of registries (snapshots) in the database.
(4) Return AL.

Transmission library represents all the logic for sending data (activities) to the server. Its functionality is the following:

- authorization of a user;
- sending data in json format.

4.4 Client Side for Linux

The Linux client was implemented in C++, which was chosen as it is an object-oriented language and provides facilities for low-level memory manipulation. SQLite was chosen as the database management system, because it is a self-contained, highly-reliable, full-featured, public-domain, serverless, transactional SQL database engine.

The Linux Agent system has three parts:

- The Measurement Tool - represents all logic for collecting and storing metrics data.
- The Sending Tool - responsible for filtering and sending metrics data to the server.
- The GUI Application or the interaction tool, which allows a user to start and stop measurement, observe the collected data, install filters, and configure settings. It is also responsible for authentication on the server and the sending thread.

Dividing the application into parts this way promotes flexibility and modifiability, and provides the possibility of applying dynamic programming. It also allows different activities like measurement collection, data transmission, GUI actions, etc. to run together without interfering with each other.

The Measurement Tool collects static measurements such as the names of the computer and the user, and information about the network. The main function of this class is to track user activity events such as FocusOut, XIKeyRelease, XIRawButtonRelease, XIButtonRelease, and XIRawButtonRelease, which are provided by the X Windows system and allow the Measurement Tool to identify the active application. After events are performed, the tool collects information about the active application: the name, ID, and pid of the application, the title of the active window, the time, the executable path, and - for browsers - the url. Most of the information is also collected by the X system. All measurements are immediately stored in the local database.

The algorithm used by the measurement tool is as follows:

(1) Determine the current application and gather all the available measurements.
(2) Subscribe to events of this application.
(3) Wait for the events.
(4) Handle the events:
   (a) FocusOut - gather the rest of metrics of application and save it. Determine the new active, focused application and begin the loop from step 1.
   (b) XIKeyRelease, XIRawButtonRelease, XIButtonRelease, XIRawButtonRelease - determine whether the internal tab or the window changed the title of the application. If there is a change - save it. Continue waiting for the events of this application from step 3.

The Sending Tool allows the user to filter the data and send the data to the main server in batch mode in JSON format. This tool also has the ability to delete data either immediately or after a delay, depending on the configuration. The filtering of the data is carried out by SQL scripts to the local database. These scripts are designed through interfaces which provide a means for extension and improvement of the filtering function. This tool also handles network connections and authorization with the server. Communication is implemented via the "curl" library. Authorization on the server is a very important process because it protects the server from receiving unauthorized data which could crash the server.

The GUI Application allows the user to control the other two modules. A user can start and stop the measuring process, the sending process, and configure the settings of these processes. The application provides a convenient way to set up the time and text filters, and allows the user to observe the data which was collected and sent. The GUI was implemented with pure X Windows calls, which is very useful for running the application on different distributions of Linux.

4.5 Dashboard Application

Having a metrics collection tool is important, but the purpose of the system is to facilitate decision making in software companies. To this end a set of process analytics and data mining tools could be devised along with the plain visual assessment of the collected

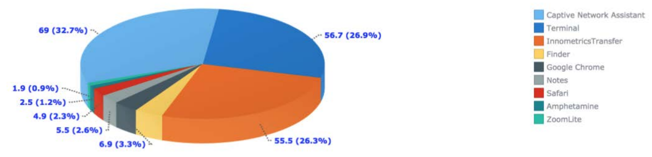

Figure 2: Sample of view for top activities from a personal dashboard.

data. A natural way to represent the collected metrics as well as the results of data analysis is visualization in the form of a dashboard.

Dashboard is an application which supports decision making by simplifying the data. Effective dashboards hide information irrelevant in certain decision making scenario. Backend part of a dashboarding application connects to a database. Frontend is rich with graphs, charts, and data visualization. A developer of dashboarding applications may have more details later, so our system should be ready to adapt to these changes.

At the moment the system implements a prototype of a flexible dashboard constructor that allows the selection of widgets and metrics in order to use them in a appropriate scenarios. The resulting dashboard is implemented in the form of a web application that communicates to the data storage, fetches, and visualizes data. In the current version of a dashboard we implemented a very simple PSP-oriented scenario for time-tracking with respect to applications used by a programmer. A sample personal dashboard of a student is presented in Fig. 2.

5 EXPERIMENTATION

In order to test the system in real environment and collect data for future analysis, it was decided to run the experimentation within a Summer Bootcamp at Innopolis University the authors are affiliated with. In class presentations were provided to recruit students who want to help us in testing and quality improvement of the Innometrics. There were two groups of master of science in software engineering students and a few groups of first year bachelor students. They were asked to go through the registration process on http://innometrics.guru:3000 portal and download agent applications corresponding to their operating system (macOS, Windows or Linux). During the period of the Bootcamp we received more than 800,000 measurements.

After students sent enough data (usually within a day or two), they were able to see collected activities on a personal web page. Participants were also able to see statistics based on their activities[4]. The Bootcamp participants were working around a week and therefore the corresponding dataset contains data coming from several independent developers. We proposed the same opportunity to the freshmen (first-year BSc students). However, the outcome was different, due to the process of agent installation on their machines. In total, we have collected 2,021,098 measurement records that describe 240,248 activities of the 23 users (12 active users work on macOS, 4 users on Linux, and 7 users on Windows).

[4]http://innometrics.guru:3000/statistics

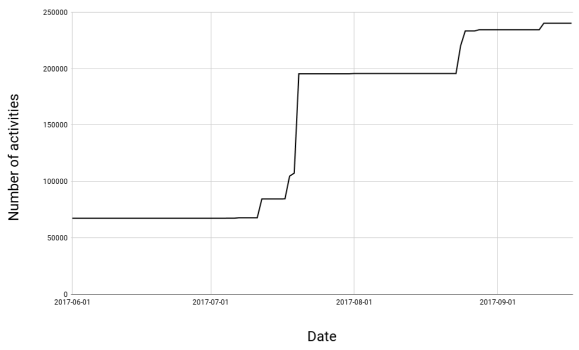

Figure 3: Data collection timeline.

Now the measurements collection system is in active usage by the developers of the Innometrics which will produce one more dataset related specifically to the development of one product. In the Fig. 3 the timeline of data collection process is represented. The first peak correspond to the Bootcamp (in the middle of July, 2017); the second peak correspond to the start of development of the second version (in late August, 2017). This data is the basis for future work on the analysis, and user's feedback, as well as bug reports, will be helpful in the system improvement.

6 DISCUSSION

The system architecture and implementation presented in this study have several features that are summarized in this section. First, the system can be adapted to heterogeneous and fluid environments that are typical in software companies. It has non-invasive agents for metrics collection on popular operating systems that developers use. The architecture of the server side is flexible with respect to possible changes of required activities and their measurements. Second, Innometrics pursues the growing trend of data privacy; it allows developers to decide which data to transmit. Finally, the system provides tools for data analysis and data visualization in real time that complies with Lean development ideas on one hand and supports decision making on the other hand.

The results of our experiments show that this system can be distributed easily in software companies to facilitate the process of measurement. Also the experiments show that installation and usage process needs deep understanding of the measurement process. Innometrics is neither a time tracker for developers, nor it can be a tool for managers to spy on developers and punish them. The primary purpose of the proposed architecture is to provide a robust tool for continuous improvement in software companies.

7 CONCLUSION AND FURTHER WORK

In this paper we have described a new approach for non-invasive software measurement systems to address some of the issues that have prevented their widespread adoption, despite they having been successfully used in localized settings [5]. The novel architecture and implementation for the non-invasive system is presented and tested in a group of university graduates. Architectural decisions behind the development of the system were justified by the requirement of high flexibility and variability of the software engineering process. The next step in our research is to verify the effectiveness of this new architecture in software companies. Additional research and development will focus on collection of source code metrics as well as the connection to requirement collection [30], to the use of third parties components [3], to the application of advanced models for data analysis as already discussed in several other works[1, 2, 4, 11, 13, 17, 23, 32–34] and the extension to distributed development [6, 7].

ACKNOWLEDGMENT

The authors would like to thank Innopolis University for generously supporting this research.

REFERENCES

[1] Pekka Abrahamsson, Raimund Moser, Witold Pedrycz, Alberto Sillitti, and Giancarlo Succi. 2007. Effort prediction in iterative software development processes–Incremental versus global prediction models. In *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on.* IEEE, 344–353.

[2] Luigi Benedicenti, Paolo Ciancarini, Franco Cotugno, Angelo Messina, Alberto Sillitti, and Giancarlo Succi. 2017. Improved Agile: A Customized Scrum Process for Project Management in Defense and Security. In *Software Project Management for Distributed Computing*. Springer International Publishing, 289–314.

[3] Justin Clark, Chris Clarke, Stefano De Panfilis, Giampiero Granatella, Paolo Predonzani, Alberto Sillitti, Giancarlo Succi, and Tullio Vernazza. 2004. Selecting components in large COTS repositories. *Journal of Systems and Software* 73, 2 (2004), 323–331.

[4] Irina Diana Coman, Alberto Sillitti, and Giancarlo Succi. 2008. Investigating the usefulness of pair-programming in a mature agile team. In *International Conference on Agile Processes and Extreme Programming in Software Engineering*. Springer Berlin Heidelberg, 127–136.

[5] Irina Diana Coman, Alberto Sillitti, and Giancarlo Succi. 2009. A Case-study on Using an Automated In-process Software Engineering Measurement and Analysis System in an Industrial Environment. In *Proceedings of the 31st International Conference on Software Engineering (ICSE 2009), Vancouver, Canada*. IEEE Computer Society, 89–99.

[6] Luis Corral, Alberto Sillitti, and Giancarlo Succi. 2012. Mobile multiplatform development: An experiment for performance analysis. *Procedia Computer Science* 10 (2012), 736–743.

[7] Luis Corral, Alberto Sillitti, Giancarlo Succi, Alessandro Garibbo, and Paolo Ramella. 2011. Evolution of mobile software development from platform-specific to web-based multiplatform paradigm. In *Proceedings of the 10th SIGPLAN symposium on New ideas, new paradigms, and reflections on programming and software.* ACM, 181–183.

[8] Enrico di Bella, Ilenia Fronza, Nattakarn Phaphoom, Alberto Sillitti, Giancarlo Succi, and Jelena Vlasenko. 2013. Pair Programming and Software Defects–A Large, Industrial Case Study. *IEEE Transactions on Software Engineering* 39, 7 (2013), 930–953.

[9] Enrico Di Bella, Alberto Sillitti, and Giancarlo Succi. 2013. A multivariate classification of open source developers. *Information Sciences* 221 (2013), 72–83.

[10] Norman E Fenton and Martin Neil. 2000. Software metrics: roadmap. In *Proceedings of the Conference on the Future of Software Engineering*. ACM, 357–370.

[11] Ilenia Fronza, Alberto Sillitti, and Giancarlo Succi. 2009. An interpretation of the results of the analysis of pair programming during novices integration in a team. In *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*. IEEE Computer Society, 225–235.

[12] Watts S Humphrey. 2005. *Psp (sm): a self-improvement process for software engineers.* Addison-Wesley Professional.

[13] Vladimir Ivanov, Manuel Mazzara, Witold Pedrycz, Alberto Sillitti, and Giancarlo Succi. 2016. Assessing the process of an Eastern European software SME using systemic analysis, GQM, and reliability growth models: a case study. In *Proceedings of the 38th International Conference on Software Engineering Companion.* ACM, 251–259.

[14] Andrea Janes, Marco Scotto, Alberto Sillitti, and Giancarlo Succi. 2006. A Perspective on Non Invasive Software Management. In *Instrumentation and Measurement Technology Conference (IMTC)*.

[15] Andrea Janes and Giancarlo Succi. 2014. Lean Software Development in Action. Springer, 187–221.

[16] Andrea A Janes and Giancarlo Succi. 2012. The dark side of agile software development. In *Proceedings of the ACM international symposium on New ideas, new paradigms, and reflections on programming and software.* ACM, 215–228.

[17] Andrejs Jermakovics, Alberto Sillitti, and Giancarlo Succi. 2011. Mining and visualizing developer networks from version control systems. In *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering.* ACM, 24–31.

[18] Philip M Johnson. 2007. Requirement and Design Trade-offs in Hackystat: An In-Process Software Engineering Measurement and Analysis System.. In *ESEM*, Vol. 7. 81–90.

[19] Philip M Johnson, Hongbing Kou, Joy Agustin, Christopher Chan, Carleton Moore, Jitender Miglani, Shenyan Zhen, and William EJ Doane. 2003. Beyond the personal software process: Metrics collection and analysis for the differently disciplined. In *Proceedings of the 25th international Conference on Software Engineering.* IEEE Computer Society, 641–646.

[20] Philip M Johnson, Hongbing Kou, Joy M Agustin, Qin Zhang, Aaron Kagawa, and Takuya Yamashita. 2004. Practical automated process and product metric collection and analysis in a classroom setting: Lessons learned from Hackystat-UH. In *Empirical Software Engineering, 2004. ISESE'04. Proceedings. 2004 International Symposium on.* IEEE, 136–144.

[21] Frank Maurer, Giancarlo Succi, Harald Holz, Boris Kötting, Sigrid Goldmann, and Barbara Dellen. 1999. Software Process Support over the Internet. In *Proceedings of the 21st International Conference on Software Engineering (ICSE '99).* ACM, 642–645. https://doi.org/10.1145/302405.302913

[22] Witold Pedrycz and Giancarlo Succi. 2005. Genetic granular classifiers in modeling software quality. *Journal of Systems and Software* 76, 3 (2005), 277–285.

[23] Witold Pedrycz, Giancarlo Succi, Alberto Sillitti, and Joana Iljazi. 2015. Data description: A general framework of information granules. *Knowledge-Based Systems* 80, Supplement C (2015), 98 – 108. https://doi.org/10.1016/j.knosys.2014.12.030 25th anniversary of Knowledge-Based Systems.

[24] Robert D Rogers and Stephen Monsell. 1995. Costs of a predictible switch between simple cognitive tasks. *Journal of experimental psychology: General* 124, 2 (1995), 207.

[25] Marco Scotto, Alberto Sillitti, Giancarlo Succi, and Tullio Vernazza. 2004. Dealing with Software Metrics Collection and Analysis: a Relational Approach. *Stud. Inform. Univ.* 3, 3 (2004), 343–366.

[26] Marco Scotto, Alberto Sillitti, Giancarlo Succi, and Tullio Vernazza. 2004. Non-invasive Product Metrics Collection: An Architecture. In *Proceedings of the 2004 Workshop on Quantitative Techniques for Software Agile Process (QUTE-SWAP '04).* ACM, New York, NY, USA, 76–78.

[27] Marco Scotto, Alberto Sillitti, Giancarlo Succi, and Tullio Vernazza. 2004. A relational approach to software metrics. In *Proceedings of the 2004 ACM symposium on Applied computing.* ACM, 1536–1540.

[28] Marco Scotto, Alberto Sillitti, Giancarlo Succi, and Tullio Vernazza. 2006. A non-invasive approach to product metrics collection. *Journal of Systems Architecture* 52, 11 (2006), 668–675.

[29] Alberto Sillitti, Andrea Janes, Giancarlo Succi, and Tullio Vernazza. 2004. Measures for mobile users: an architecture. *Journal of Systems Architecture* 50, 7 (2004), 393–405.

[30] Alberto Sillitti and Giancarlo Succi. 2005. Requirements engineering for agile methods. In *Engineering and Managing Software Requirements.* Springer Berlin Heidelberg, 309–326.

[31] Alberto Sillitti, Giancarlo Succi, and Stefano De Panfilis. 2006. Managing non-invasive measurement tools. *Journal of Systems Architecture* 52, 11 (2006), 676–683.

[32] Alberto Sillitti, Giancarlo Succi, and Jelena Vlasenko. 2012. Understanding the impact of pair programming on developers attention: a case study on a large industrial experimentation. In *Proceedings of the 34th International Conference on Software Engineering.* IEEE Press, 1094–1101.

[33] Giancarlo Succi, Luigi Benedicenti, and Tullio Vernazza. 2001. Analysis of the effects of software reuse on customer satisfaction in an RPG environment. *IEEE Transactions on Software Engineering* 27, 5 (2001), 473–479.

[34] Giancarlo Succi, James Paulson, and Armin Eberlein. 2001. Preliminary results from an empirical study on the growth of open source and commercial software products. In *EDSER-3 Workshop.* 14–15.

[35] Alejandro Vera-Baquero, Ricardo Colomo-Palacios, and Owen Molloy. 2013. Business process analytics using a big data approach. *IT Professional* 15, 6 (11 2013), 29–35.

[36] Tullio Vernazza, Giampiero Granatella, Giancarlo Succi, Luigi Benedicenti, and Martin Mintchev. 2000. Defining metrics for software components. In *5th World Multi-Conference on Systemics, Cybernetics and Informatics, Florida,* Vol. 11. 16–23.