www.sensorsportal.com

Sensors & Transducers
Published by IFSA Publishing, S. L., 2020
http://www.sensorsportal.com

The Usage of Process Metrics to Analyze the Energy Efficiency of the Software Development Process and Product

Daniel ATONGE, Shokhista ERGASHEVA, * Artem KRUGLOV, Dragos STRUGAR, Andrey SADOVYKH, Giancarlo SUCCI, Xavier VASQUEZ and Evgeny ZOUEV
Innopolis University, Universitetskaya st., 1, 420500 Innopolis, Russia
* E-mail: a.kruglov@innopolis.ru

Received: 16 July 2020 /Accepted: 26 August 2020 /Published: 30 September 2020

Abstract: The InnoMetrics project aims at building and validating a quantitative framework to assess and guide the software development teams using process metrics collected non-invasively throughout the life-cycle of software systems, from the initial concept to the deployment, execution, and maintenance taking into consideration energy concerns, which play a pivotal role in the success of applications and infrastructures. In this paper, we report the early experience we have in its development together with the data of developers' activities that we have obtained so far, including running processes and applications, user actions in browser or IDE, and associated energy consumption.

Keywords: Non-invasive measurement, Granular computation, Energy efficiency, Software development, Process metrics.

1. Introduction

With the growing need for IT devices, the pervasiveness of energy efficiency is rising rapidly. According to the data from U.S. Energy Information Administration [1], global resource consumption began exceeding planetary supply by 2030, and the consumption of energy is increased more than three times starting from 1980 till 2018. To be concise, the world consumed about 23000 billion kWh in 2017. These kinds of numbers let us know that the energy consumption issue is becoming much more globalized.

If we look at the daily behavior of people related to energy consumption, there are lots of habits been neglected by us, where we do not even think about it. For example, there is a tendency of leaving the PC running while switching between the jobs. The running computer, estimating a typical desktop PC with a 17-inch LCD monitor, consumes electricity about 65-100 watts and 35 watts for the computer and monitor respectively. If the computer system is left on 24/7 for a year, it will consume 874 kilowatt-hours of electricity that is enough to release 341 kg of carbon dioxide into the atmosphere and can also be equated to driving 1312 km in an average car [2].

Keeping in mind the importance of the above-mentioned problems, many organizations have started to design their products in the way that they will consume less energy not only in terms of financial expense limitation but to minimize the heat emission and support a green environment. The process of energy consumption analysis, which becomes one of the major issues for any software development project today, defines the processes that require the power the most.

An important role in the design of energy-efficient systems takes the analysis of values of the basic parameters (metrics) of energy consumption. Software metrics are quantitative measures of specific attributes of software development, including software process, product, and resource metrics [3-4]. There are several kinds of product metrics, based on the analysis of source code, developed during the past few decades for different programming paradigms such as structured programming and object-oriented programming (OOP). To provide a novelty approach to energy efficiency assessment of the software product, we focused on the development process analysis metrics [4]. However, the group of process metrics has not properly been observed yet [5]. One of the reasons for this is the absence of a tool for sufficient analysis of the development process [4].

Usually, such a process involves the participation of the developer which is called an invasive measurement collection method [6], where we faced the problem of subjective measurement. Furthermore, it in turn leads to the increase in time costs of the project as far as it requires the personal involvement of the developers where switching between tasks can be a disturbing and time-consuming activity. Thus, software metrics collection in a non-invasive way - where metrics are collected automatically, without developers' intervention - is claimed as promising approaches in this area [7-9]. This approach allows us to track a variety of software development process factors affecting its efficiency and calculate it in real-time. The advantages of this approach can be

- The process is analyzed continuously and not on a punctual basis;
- The granularity of the data derived can be maximized opposed to invasive metric collection method;
- The process itself will proceed without interrupting the developers from the main workflow, hence the data can be collected more reliably [4].

Moreover, the toolkit integrates with the most used software development environment and office applications. Development of the framework which provides a non-invasive way of collecting software development process metrics could result in a set of vital metrics and development effort patterns. Using effective visualization of the results of data analysis, one can get sufficient insights into the development process and its energy efficiency.

The framework we are proposing, InnoMetrics, is mainly focused on energy consumption metric collection [10] and analysis of software development process improvement based on the insights derived from the framework. InnoMetrics collects the data throughout the development process automatically and sends the data to the server automatically in an established time interval by the developer for further analysis. It allows the developers and managers to be aware of the software development process's strengths and weaknesses and which is more about the energy consumption of the process itself and the developing product. By virtue of the data, its visualized representation concerning metric and analysis gives a decisive vision of the overall process in real-time. As a benefit of the framework can serve the modifiability of the framework depending on the size and latitude of the company and the development team.

2. Energy Metrics

The system InnoMetrics is basically developed based on the monitoring of the software development process energy efficiency and the developers' teams productivity. As the Systematic Literature Review (SLR) we did at the beginning of the project resulted in the set of different energy-related metrics of the development process [4]. Throughout the study, metrics were divided into three types: process, product, and code (hardware).

In general, all studies in SLR were devoted to real measurement and model-based measurement. As this kind of measurement involves usage of third-party hardware tools to get energy metrics from various components, we considered them as out of the scope of our non-invasive software development process analysis approach. The study suggests that code analysis is thoroughly analyzed, nonetheless, the group of process metrics was not properly explored. The main reason for this insufficient analysis of process measurements is the absence of the tool. Furthermore, the process cost increases since usually it requires the developers' participation.

The energy consumption of software applications was thoroughly researched and concluded based on the research findings.

Battery draining applications result in lower user experience and dissatisfied users. Optimal battery usage (energy usage) is an important aspect that every client must consider.

Application energy consumption is dependent on a wide variety of system resources and conditions. Energy consumption depends on, but is not limited to, the processor the device uses, memory architecture, the storage technologies used, the display technology used, the size of the display, the network interface that you are connected to, active sensors, and various conditions like the signal strength used for data transfer, user settings like screen brightness levels, and many more user and system settings.

For precise energy consumption measurements, one needs specialized hardware. While they provide the best method to accurately measure energy consumption on a particular device, such a methodology is not scalable in practice, especially if such measurements have to be made on multiple devices. Even then, the measurements by themselves will not provide much insight into how the application contributes to the battery drainage, making it hard to focus on any application optimization efforts.

The InnoMetrics system aims at enabling users to estimate their application's energy consumption without the need for specialized hardware. Such estimation is made possible using a software power model that has been trained on a reference device representative of the low powered devices applications might run on.

2.1. Windows Energy Metrics

Based on the findings of the research, metrics like following were investigated [5]:

- Software Energy Consumption (SEC) - the total energy consumed by the software;

- Unit Energy Consumption (UEC) - the energy consumed by a specific unit of the software;

Considering our profiling method and the tools available for us, the ability to attribute the energy consumption was possible only at the process level in coarse granularity. However, the hardware resource usage can fill the gap when it comes to accurately relating Energy Consumption (EC) to individual software elements hence enabling the computation of the UEC.

Profiling the performance requires a basic understanding of hardware components that has to be monitored through "performance counters", which is possible in Windows System. While interpreting performance data for further analysis, the context information has to be taken into account (e.g. hardware-specific details).

To evaluate the Unit Energy Consumption (UEC) the following hardware resources should be monitored:

- Hard disk: disk bytes/sec, disk read bytes/sec, disk write bytes/sec;

- Processor: percentage of processor usage;

- Memory: private bytes, working set, private working set;

- Network: bytes total/sec, bytes sent/sec, bytes received/sec;

- IO: IO data (bytes/sec), IO read (bytes/sec), IO write (bytes/sec).

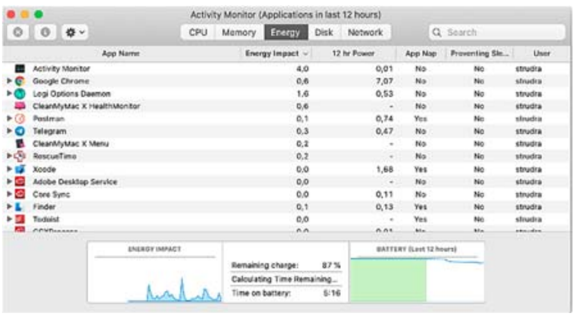
Attributing some weights to elements of the UEC or by some reliable assumption such as considering the power model to be linear in nature for each individual component, the SEC Metric is computed.

Besides, the energy usage can also be appraised using Performance Counter, Performance Counter Category, and related classes that are available with .NET Framework [11-12]. To be specific, by analyzing the MSDN documentation to attain the goal of collecting energy-related metrics, it was concluded that the information about CPU time, Total Processor Time per process, CPU usage, Memory usage, network usage that Performance Counter provides can be reliable.

The bottleneck in this situation is that it is difficult to match up constantly changing application process IDs and names. The energy consumption of the system depends on a variety of factors that are not limited to those which can be collected using the above-mentioned performance classes.

2.2. MacOS Energy Metrics

In order to obtain energy-related data, we explored some of the APIs that MacOS provides. The first and most obvious tool was Activity Monitor, an application that comes built-in every MacOS system. One of its aspects is energy consumption, shown in Fig. 1.



App Name	Energy Impact	12 hr Power	App Nap	Preventing Sle...	User
Activity Monitor	4,0	0,01	No	No	strudra
Google Chrome	0,6	7,07	No	No	strudra
Logi Options Daemon	1,6	0,53	No	No	strudra
CleanMyMac X HealthMonitor	0,6	-	No	No	strudra
Postman	0,1	0,74	Yes	No	strudra
Telegram	0,3	0,47	No	No	strudra
CleanMyMac X Menu	0,2	-	No	No	strudra
RescueTime	0,2	-	No	No	strudra
Xcode	0,0	1,68	Yes	No	strudra
Adobe Desktop Service	0,0	-	No	No	strudra
Core Sync	0,0	0,11	No	No	strudra
Finder	0,1	0,13	Yes	No	strudra
Todoist	0,0	-	Yes	No	strudra

Remaining charge: 87 %
Calculating Time Remaining...
Time on battery: 5:16

Fig. 1. MacOS energy metrics.

The second column, Energy Impact, attracted our attention immediately. We then wanted to figure out what these values (4.0, 0.6, 1.6, etc.) represent. It has been then brought to our attention that the definition of Energy Impact is not precisely defined by Apple. According to Activity Monitor's documentation, the definition of Energy Impact is *"A relative measure of the current energy consumption of the app. Lower numbers are better"* [13]. In another document [14] it is stated that *"The Energy tab of Activity Monitor displays the Energy Impact of each open app based on a number of factors including CPU usage, network traffic, disk activity and more. The higher the number, the more impact an app has on battery power"*. Both of these are vague, and we needed a concrete way of obtaining this metric's values.

One article on the Mozilla blog attempted to figure out a formula for this. They indicated that the result of macOS's `top` command-line tool which performs periodic measurements of all kinds of metrics; including ones relevant to energy consumption: CPU usage, wakeups, and the power measure. The article we mentioned above suggests running the following command to get the above-mentioned information:

```
top  -stats  pid,command,cpu,idlew,power  -o
power -d
```

To yield the results (trimmed) as given in Table 1.

Table 1. Top results.

PID	COMMAND	% CPU	IDLEW	POWER
50300	Firefox	12.9	278	26.6
76256	Plugin-container	3.4	159	11.3
151	Coreaudiod	0.9	68	4.3
76505	Top	1.5	1	1.6
76354	Activity Monitor	1.0	0	1.0

They go on to suggest that POWER measure is calculated using a simple formula, and a specific configuration file (tuned for every machine's architecture). Using the findings from that article, we decided to use some of the most impactful metrics:

- Battery percentage;
- Battery status (is charging or not);
- RAM measurement (how much RAM does the active process use);
- vRAM measurement (how much vRAM does the active process occupy);
- % CPU utilized (per process).

All of these metrics were obtained using the macOS command-line interface. E.g. to get the current battery percentage we used `pmset -g batt`, and for other measurements, we used the `ps -axm -o` command with varying parameters (depending on the use case). It was also possible to use the top command, but as we are performing periodic checks anyway, `the top` was not necessary. Further investigation on the impact of these metrics, as well as some others, is a crucial part of our research agenda.

3. System Description

The framework for the non-invasive approach for software metrics collection and analysis consists of three parts, which were defined in a high-level of abstraction of the general architecture from the logical point of view (see Fig. 2.):

- Data Collectors, for collecting data from different OS types;
- Server, which includes the analytic module for quantitative and qualitative analysis of the obtained data;
- Dashboard, for the visual representation of information about the development process.

In general, the architecture has 3 main parts mentioned above, however, the detailed information of all components separately will be described throughout this section.

The data collectors in the architecture are a set of services developed for major operating systems, in which they have the main aim of monitoring the activities that users perform on their devices, and collect the data needed to calculate and analyze the energy consumption of the device under usage and the process efficiency.

DataCollector API is the main point of interaction for the data collection components with the centralized database, in the same way, this API handles the outgoing notifications to these components.

Innometrics Database, the information repository in which the raw information collected is stored, which will be transformed and analyzed through the Analytics service.

Analytic service is an automatic data transformation process, whose main purpose is to take the raw collected data and transform it to perform a much faster analysis process, without impacting the performance of the transactional database. Whereas in InnoMetrics Analytics Database, the analysis of the information collected is carried out, which will not have the information in real-time, but a periodic load of the information collected and transformed will be performed. Finally, the Dashboard backend component provides an interface between the presentation layer and the analysis database.

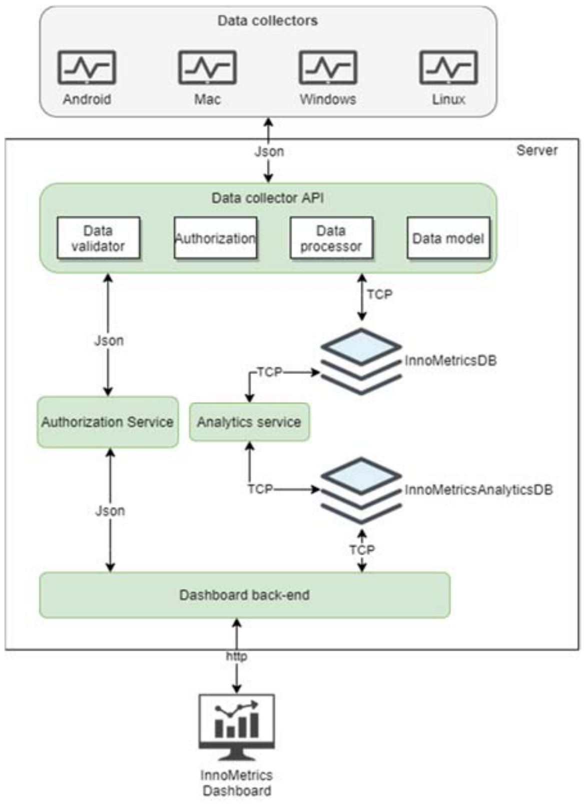


Fig. 2. System architecture. Logical view.

The interface of the system is a small control module where the users are in charge of a specific project have the ability to visualize information with the help of graphs, charts, and maps. InnoMetrics Dashboard is a web system, focused on visualizing the analyzed information based on the obtained data from collectors during the development process. The analyzed data can play a vital role in the Agile software development process optimization which helps in the decision-making process. In addition, it will have a small module to manage the system settings like automatic data transfer based on the time interval the user set, the users' ability to send error reports to the developers, and ease of collected metrics modification.

Before analysis of the application design methods, the set of functional and non-functional requirements were established in terms of the system implementation. These requirements and the constraints under which the system should operate and be developed have distinctive features from other software metrics collection systems. The main functionality that our system defines is the energy-related metrics collection and analysis. Based on these process metrics, the energy efficiency prediction and development process optimization can be of paramount importance.

In order to provide quality attributes like scalability, fault-tolerance, and adaptability to new requirements of the system, it is based on a micro-services architecture with two main access points and a series of specialized services such as Analytic services, Administration services, Authorization service, etc.

For an initial deployment of the solution, there is a server that will host all the services (see Fig. 3).

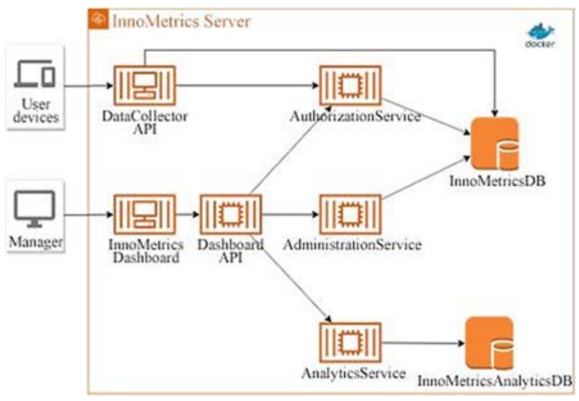


Fig. 3. System architecture. Physical view.

The user devices here in the figure are tracked by the data collection components that are only focused on data collection, storage and transmission. While Managers component represents the user with a manager role of each project to where they have access to a dashboard where they can easily monitor the performance of their work team from the information collected.

The DataCollector service is mainly in charge of providing an interface of communication for the different data collectors so they can store data in our data repository. Additionally, providing a notification interface to send information from the core system to these external components.

AuthorizationService component specializes in the authentication and authorization of users, generation, and validation of tokens that are required to process any request that is processed by the different services within the system.

DashboardAPI is responsible for providing the necessary data to generate graphics within the dashboard. AdministrationService provides the functionalities that allow you to make configurations within the system, such as user and role administration, configure the frequency with which the data analysis process is executed, among others.

Administration Service provides the functionalities that allow the users to make configurations within the system, such as user and role administration, configure the frequency with which the data analysis process is executed, etc.

AnalyticsService is in charge of carrying out the data transformations to generate the information required by end-users.

With the purpose to provide an agile way to deploy and scale the system, Docker container is used as a virtualization engine, having OpenJDK as the main JVM of the services and Postgres as RDBMS.

The diagram in Fig. 4 provides a high-level perspective on the implementation of the API used by the data collector components, which is being developed under the Spring Boot framework. It incorporates such functionalities as Spring boot security for user validation processes and authorization, Spring boot Data as a persistence engine. Additionally, there is the support of third-party libraries that will be described later.

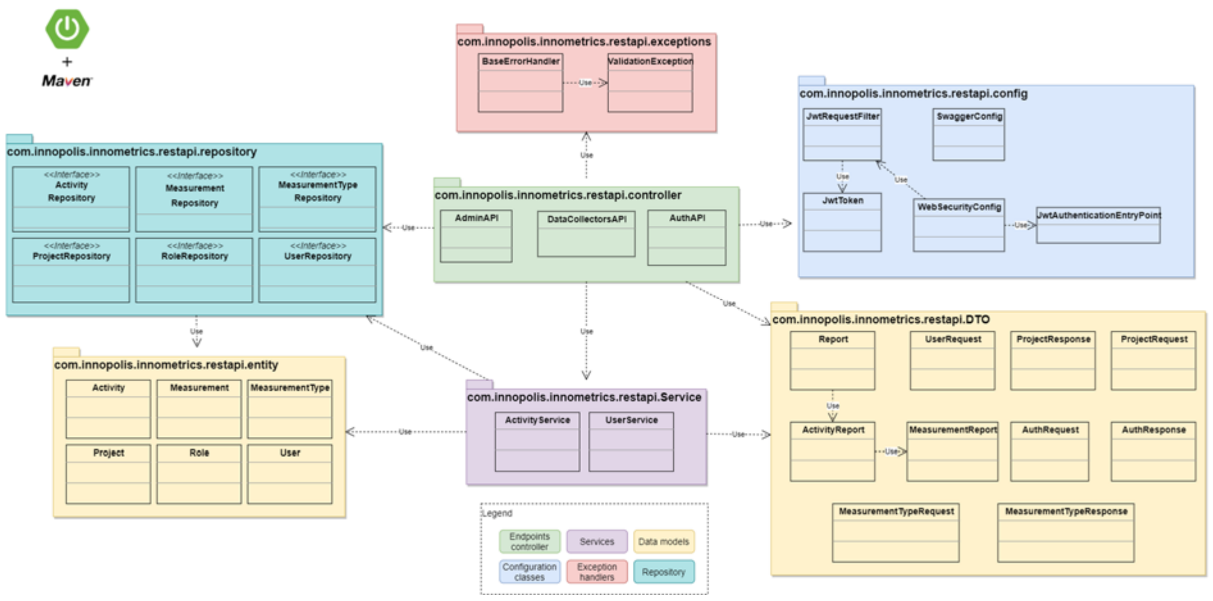

+
Maven
com.innopolis.innometrics.restapi.exceptions
BaseErrorHandler
ValidationException
com.innopolis.innometrics.restapi.config
JwtRequestFilter
SwaggerConfig
JwtToken
WebSecurityConfig
JwtAuthenticationEntryPoint
com.innopolis.innometrics.restapi.repository
<<interface>> Activity Repository
<<interface>> Measurement Repository
<<interface>> MeasurementType Repository
<<interface>> ProjectRepository
<<interface>> RoleRepository
<<interface>> UserRepository
com.innopolis.innometrics.restapi.controller
AdminAPI
DataCollectorsAPI
AuthAPI
com.innopolis.innometrics.restapi.entity
Activity
Measurement
MeasurementType
Project
Role
User
com.innopolis.innometrics.restapi.Service
ActivityService
UserService
com.innopolis.innometrics.restapi.DTO
Report
UserRequest
ProjectResponse
ProjectRequest
ActivityReport
MeasurementReport
AuthRequest
AuthResponse
MeasurementTypeRequest
MeasurementTypeResponse
Use
Legend
Endpoints controller
Services
Data models
Configuration classes
Exception handlers
Repository

Fig. 4. System technical architecture.

Within the *controller* package are the classes to be exposed as REST services, which are divided into 3 classes:

- *AdminAPI* exposes those methods that are related to the administration of users, roles and system configurations;
- *AuthAPI* provides the mechanisms for user authentication and token generation;
- *DataCollectorAPI*, the controller in charge of exposing the methods that are used by the different collection modules for the transmission of data to the central repository.

Following the structure proposed in the spring boot architecture, the services layer contains the business logic needed in each of the processes exposed by the controllers and also provides an abstraction layer between the controllers that make use of DTO (Data Transfer Objects) and the persistence layer that naturally maps the entities contained in the database.

Using the functionalities provided by the implementation of JPA in spring boot through the *JPARepository* interface, this layer contains the persistence interfaces of the entities modeled within the entity package.

In the *entity* package, we have the models of the entities that are involved in the data collection process, which are an abstraction of the database entities.

The *DTO* layer models the objects used in the request and response processes of each method exposed in the controller layer, this layer allows us to decouple the data collection components and the model that is being persisted in the database.

Within *the config* layer are those classes that provide the configuration mechanisms for the libraries used, such as, spring boot security configuration to integrate it with the JsonWebToken (JWT) library, which is in charge of the generation of authentication tokens. As well as the configuration spring fox-swagger libraries for the API documentation generation.

Exception is a layer that allows us to make an extension of the RuntimeException class to provide a unified exceptions handling mechanism.

4. Data Collection Process

Data collection from the users' devices is described in detail in Fig. 5, to be able to store the information collected within the central repository.

It is suggested that the components perform the data load process periodically but not in real-time because of the technical limitation in terms of real-time data collection API. Each of the requests must include an authorization token which was decided to establish as 90 days in order not to overload the users every time with the authentication process. Nevertheless, if the token is not valid or is expired, the component that performs the request will be notified with an error response. Under such conditions, the component has to request a new token or request the user access credentials.

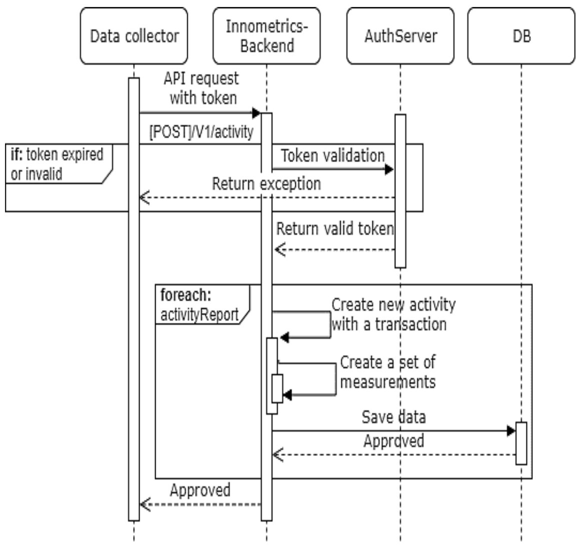


Fig. 5. Data collection sequence diagram.

Then, the API loads the data sequentially and returns a flag in each of the activities sent in order to notify the collection component of the status of the load. The same process follows in order to provide the components with the ability to re-try the load of information that could not be processed and notify the user about these problems.

As a result of the above-mentioned sequence of activities, the following data is being collected in tabular form: process name, process id, status (app focus or idle), start time, end time, IP address, mac address, process description, battery power, memory and GPU utilization.

Further, at the hand of these data collected, the analysis of the process is done. The time data of the development process such as start time and end time, program status are used to calculate the time spent for particular programs during the given period of time.

5. Implementation

The collector is presently an application with the following interfaces:

- Registration interface for the users;
- Login interface for the users;
- Collector Interface: Which displays data collected from the host's machine.

5.1. Registration Interface

By using this interface if the user is not signed in yet, the user has either login or create a new account of which sequence of actions is described in Fig. 6. If the user chooses to create an account, the parameters like token and user details like email, name, surname, and password should be included via the provided interface.

The user will be provided with one of the following responses as feedback (Table 2).

Table 2. Feedback types as a response.

200	OK
201	Created
401	Unauthorized
403	Forbidden
404	Not Found

5.2. Login Interface

If the user isn't signed in and wants to send the collected data to the backend at the end of the working day, the only option is first to login with the user credentials and then send. The required parameter to be provided while login in Authentication Request which contains the information about email, password, and project ID fields. Otherwise, the data collected has a defined time interval that is set to send the data collected automatically without user intervention. The whole sequence of process activities is illustrated in Fig. 7 below.

The feedback to the AuthenticationRequest as responses are described in Table 2.

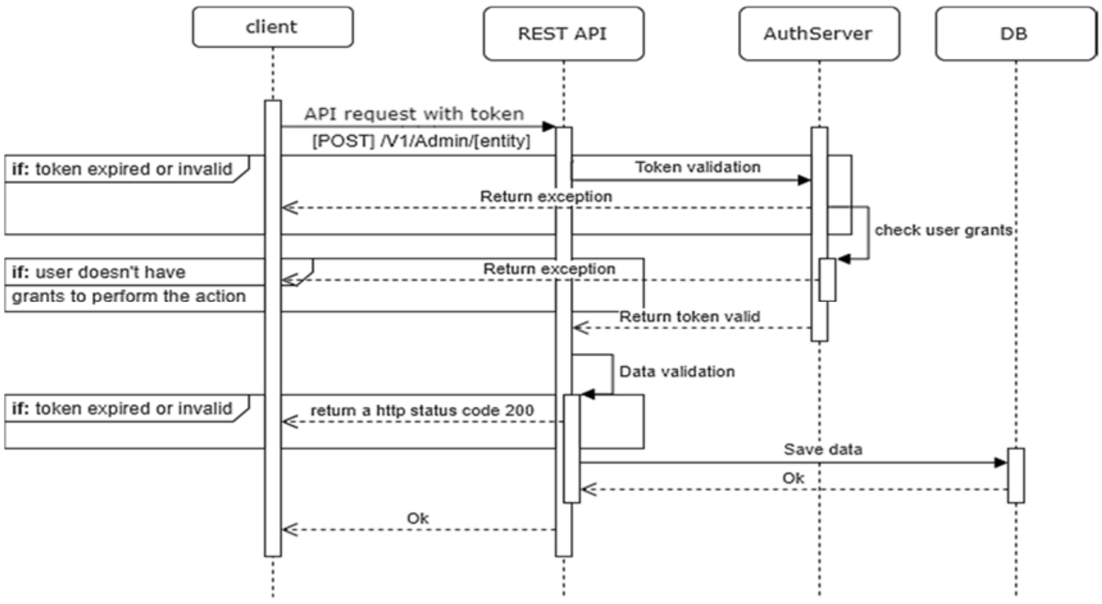

client
REST API
AuthServer
DB
API request with token
[POST] /V1/Admin/[entity]
if: token expired or invalid
Token validation
Return exception
check user grants
if: user doesn't have grants to perform the action
Return exception
Return token valid
Data validation
if: token expired or invalid
return a http status code 200
Save data
Ok
Ok

Fig. 6. User registration sequence diagram.

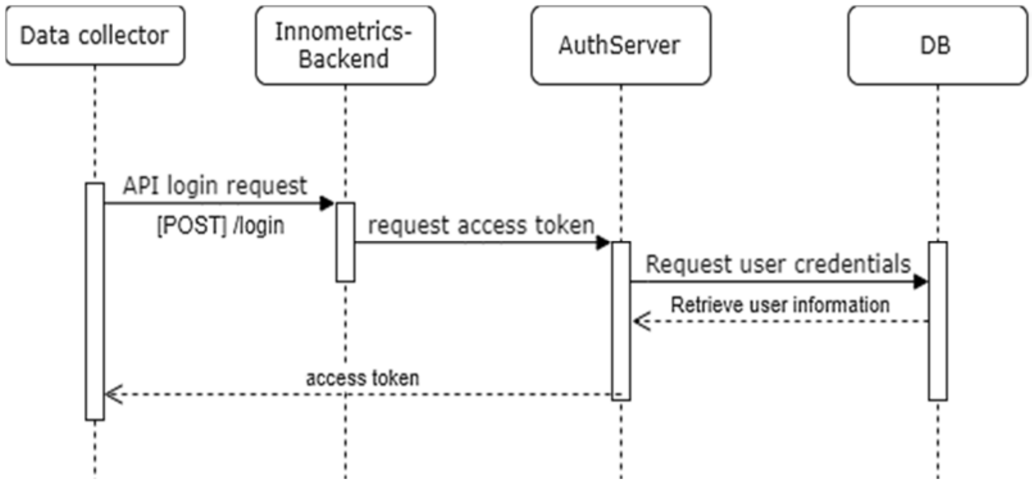


Fig. 7. User login sequence diagram.

5.3. Collector Interface

The parameters to send the collected data to the back-end should include the token and the report. Here, the token should be added to the header. The report structure should be as following:

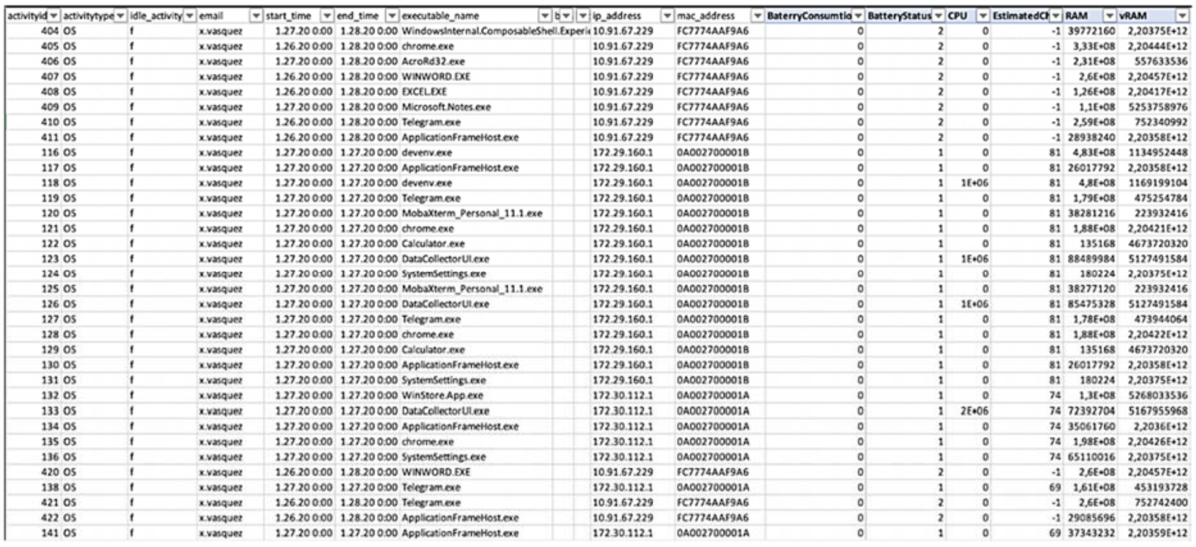
```
{
"activities": [
 {
 "activityID": 0,
 "activityType": "string",
 "browser_title": "string",
 "browser_url": "string",
"end_time":"2019-11-28T09:30:38.470Z",
 "executable_name": "string",
 "idle_activity": true,
 "ip_address": "string",
 "mac_address": "string",
 "measurements": [
 {
 "measurementTypeId": 0,
 "value": 0
 }
 ],
"start_time":"2019-11-28T09:30:38.471Z",
"userID": "string"
 }
]
}
```

Here in the data collector interface, the minimal interaction will be possessed, as sending the data collected is triggered by just one click (send). Where token and the report of collected data with required parameters are sent to the backend. The required parameters are collected as discussed in the previous section (see Fig. 5.). The response feedback is described in Table 1. If the users' local database isn't empty, the data collected will be transferred to the backend via a REST API.

6. Testing

As soon as the development of the system has been finished, the beta testing phase was arranged with the contribution of university graduates. During the beta testing phase of our system, we have been able to collect data about 50+ applications used by the participants of the test group, with a daily average of 2500 samples.

After testing our data collector for a long period of time, and running queries in the back-end to export collected data into CSV format, we obtain the following result (see Fig. 8). Added to the previous system test, this validates that our developed collector successfully communicates with the back-end without loss of data.



activityid	activitytype	idle_activity	email	start_time	end_time	executable_name	b		ip_address	mac_address	BaterryConsumtio	BatteryStatus	CPU	EstimatedCh	RAM	vRAM
404	OS	f	x.vasquez	1.27.20 0:00	1.28.20 0:00	WindowsInternal.ComposableShell.Experi			10.91.67.229	FC7774AAF9A6	0	2	0	-1	39772160	2,20375E+12
405	OS	f	x.vasquez	1.26.20 0:00	1.28.20 0:00	chrome.exe			10.91.67.229	FC7774AAF9A6	0	2	0	-1	3,33E+08	2,20444E+12
406	OS	f	x.vasquez	1.27.20 0:00	1.28.20 0:00	AcroRd32.exe			10.91.67.229	FC7774AAF9A6	0	2	0	-1	2,31E+08	557633536
407	OS	f	x.vasquez	1.26.20 0:00	1.28.20 0:00	WINWORD.EXE			10.91.67.229	FC7774AAF9A6	0	2	0	-1	2,6E+08	2,20457E+12
408	OS	f	x.vasquez	1.26.20 0:00	1.28.20 0:00	EXCEL.EXE			10.91.67.229	FC7774AAF9A6	0	2	0	-1	1,26E+08	2,20417E+12
409	OS	f	x.vasquez	1.27.20 0:00	1.28.20 0:00	Microsoft.Notes.exe			10.91.67.229	FC7774AAF9A6	0	2	0	-1	1,1E+08	5253758976
410	OS	f	x.vasquez	1.26.20 0:00	1.28.20 0:00	Telegram.exe			10.91.67.229	FC7774AAF9A6	0	2	0	-1	2,59E+08	752340992
411	OS	f	x.vasquez	1.26.20 0:00	1.28.20 0:00	ApplicationFrameHost.exe			10.91.67.229	FC7774AAF9A6	0	2	0	-1	28938240	2,20358E+12
116	OS	f	x.vasquez	1.27.20 0:00	1.27.20 0:00	devenv.exe			172.29.160.1	0A002700001B	0	1	0	81	4,83E+08	1134952448
117	OS	f	x.vasquez	1.27.20 0:00	1.27.20 0:00	ApplicationFrameHost.exe			172.29.160.1	0A002700001B	0	1	0	81	26017792	2,20358E+12
118	OS	f	x.vasquez	1.27.20 0:00	1.27.20 0:00	devenv.exe			172.29.160.1	0A002700001B	0	1	1E+06	81	4,8E+08	1169199104
119	OS	f	x.vasquez	1.27.20 0:00	1.27.20 0:00	Telegram.exe			172.29.160.1	0A002700001B	0	1	0	81	1,79E+08	475254784
120	OS	f	x.vasquez	1.27.20 0:00	1.27.20 0:00	MobaXterm_Personal_11.1.exe			172.29.160.1	0A002700001B	0	1	0	81	38281216	223932416
121	OS	f	x.vasquez	1.27.20 0:00	1.27.20 0:00	chrome.exe			172.29.160.1	0A002700001B	0	1	0	81	1,88E+08	2,20421E+12
122	OS	f	x.vasquez	1.27.20 0:00	1.27.20 0:00	Calculator.exe			172.29.160.1	0A002700001B	0	1	0	81	135168	4673720320
123	OS	f	x.vasquez	1.27.20 0:00	1.27.20 0:00	DataCollectorUI.exe			172.29.160.1	0A002700001B	0	1	1E+06	81	88489984	5127491584
124	OS	f	x.vasquez	1.27.20 0:00	1.27.20 0:00	SystemSettings.exe			172.29.160.1	0A002700001B	0	1	0	81	180224	2,20375E+12
125	OS	f	x.vasquez	1.27.20 0:00	1.27.20 0:00	MobaXterm_Personal_11.1.exe			172.29.160.1	0A002700001B	0	1	0	81	38277120	223932416
126	OS	f	x.vasquez	1.27.20 0:00	1.27.20 0:00	DataCollectorUI.exe			172.29.160.1	0A002700001B	0	1	1E+06	81	85475328	5127491584
127	OS	f	x.vasquez	1.27.20 0:00	1.27.20 0:00	Telegram.exe			172.29.160.1	0A002700001B	0	1	0	81	1,78E+08	473944064
128	OS	f	x.vasquez	1.27.20 0:00	1.27.20 0:00	chrome.exe			172.29.160.1	0A002700001B	0	1	0	81	1,88E+08	2,20422E+12
129	OS	f	x.vasquez	1.27.20 0:00	1.27.20 0:00	Calculator.exe			172.29.160.1	0A002700001B	0	1	0	81	135168	4673720320
130	OS	f	x.vasquez	1.27.20 0:00	1.27.20 0:00	ApplicationFrameHost.exe			172.29.160.1	0A002700001B	0	1	0	81	26017792	2,20358E+12
131	OS	f	x.vasquez	1.27.20 0:00	1.27.20 0:00	SystemSettings.exe			172.29.160.1	0A002700001B	0	1	0	81	180224	2,20375E+12
132	OS	f	x.vasquez	1.27.20 0:00	1.27.20 0:00	WinStore.App.exe			172.30.112.1	0A002700001A	0	1	0	74	1,3E+08	5268033536
133	OS	f	x.vasquez	1.27.20 0:00	1.27.20 0:00	DataCollectorUI.exe			172.30.112.1	0A002700001A	0	1	2E+06	74	72392704	5167955968
134	OS	f	x.vasquez	1.27.20 0:00	1.27.20 0:00	ApplicationFrameHost.exe			172.30.112.1	0A002700001A	0	1	0	74	35061760	2,2036E+12
135	OS	f	x.vasquez	1.27.20 0:00	1.27.20 0:00	chrome.exe			172.30.112.1	0A002700001A	0	1	0	74	1,98E+08	2,20426E+12
136	OS	f	x.vasquez	1.27.20 0:00	1.27.20 0:00	SystemSettings.exe			172.30.112.1	0A002700001A	0	1	0	74	65110016	2,20375E+12
420	OS	f	x.vasquez	1.26.20 0:00	1.28.20 0:00	WINWORD.EXE			10.91.67.229	FC7774AAF9A6	0	2	0	-1	2,6E+08	2,20457E+12
138	OS	f	x.vasquez	1.27.20 0:00	1.27.20 0:00	Telegram.exe			172.30.112.1	0A002700001A	0	1	0	69	1,61E+08	453193728
421	OS	f	x.vasquez	1.26.20 0:00	1.28.20 0:00	Telegram.exe			10.91.67.229	FC7774AAF9A6	0	2	0	-1	2,6E+08	752742400
422	OS	f	x.vasquez	1.26.20 0:00	1.28.20 0:00	ApplicationFrameHost.exe			10.91.67.229	FC7774AAF9A6	0	2	0	-1	29085696	2,20358E+12
141	OS	f	x.vasquez	1.27.20 0:00	1.27.20 0:00	ApplicationFrameHost.exe			172.30.112.1	0A002700001A	0	1	0	69	37343232	2,20359E+12

Fig. 8. Data stored on the back-end.

Based on the information we collect on a daily basis, the development team is in charge of analyzing and assessing some relevant aspects, quality in the data collection process, and data itself. Which will help us to develop a reliable and consistent data model, to subsequently move on to the next stage - experiment on the energy consumption metrics collection. Besides, starting focusing on deep data analysis and infer additional information on energy efficiency from collected data will be the main concentration**.**

7. Conclusions

In this paper the new approach of measuring the software development process metrics in a non-invasive way. The architecture and implementation of this system were announced and tested with the contribution of university students. For now, all the necessary sorts of operating system versions were developed for collecting the basic process metrics mentioned above. However, for some platforms, it is prohibited to collect such information about user processes at OS level due to security reasons [14]. Thus, the extraction of reliable energy consumption data at the required level of granularity is the aspect of further investigation.

We came up with using process metrics in combination with product metrics to reconstruct the development process.

Architectural decisions of the development were justified with the non-functional requirements we are focused on. Based on the requirements and research on energy consumption process metrics, additional metrics will be added to the system to collect for more reliable results. The next step in our research is to verify the results of the collected data from industrial companies.

In addition, we will focus on energy metrics collection for different data collectors and other agents like software management systems and integrated development environments in future work.

Acknowledgments

This research project is carried out under the support of the Russian Science Foundation Grant No 19-19-00623.

References

[1]. Energy information administration. viewed: July 10, 2020: https://www.eia.gov/international/overview/world

[2]. Green facts. viewed: July 10, 2020 https://www.greenlivingpedia.org/Green_facts

[3]. Jagroep E., *et al*., An energy consumption perspective on software architecture, *Software Architecture, Issue 9278 in LNCS*, *Springer*, 2015, pp. 239-247.

[4]. Shokhista Ergasheva, *et al*., Metrics of Energy Consumption in Software Systems: A Systematic Literature Review, in *Proceedings of the 3rd International Conference on Power and Energy Engineering,* 2019.

[5]. Jagroep E., *et al*., Energy efficiency on the product roadmap: an empirical study across releases of a software product, *Journal of Software: Evolution and Process,* 2016, pp. 1-34.

[6]. Hayri A., *et al*., The Impact of Source Code in Software on Power Consumption, *International Journal of Electronic Business Management*, Electronic Business Management Society, Taiwan, Vol. 14, 2016, pp. 42-52.

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

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

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

[10]. Artem Kruglov, Daniel Atonge, Dragos Strugar, Giancarlo Succi, Shokhista Ergasheva, Xavier Vasquez, Software Development Analysis for Energy Efficiency Using Process Metrics, in *Proceedings of the 2nd International Conference on Advances in Signal Processing and Artificial Intelligence (ASPAI'2020)*, Berlin, Germany, 2020, pp. 175-181.

[11]. Performance counters. https://docs.microsoft.com/en-us/windows/win32/perfctrs/performance-counters-portal, December 2019.

[12]. Performance counters in the .NET framework. https://docs.microsoft.com/en-us/dotnet/framework/debug-trace-profile/performance-counters, December 2019.

[13]. Apple Inc. 2020, How to use Activity Monitor on your Mac, viewed: February 21, 2020 https://support.apple.com/en-au/HT201464.

[14]. Apple Inc. 2020, About Mac Notebook Batteries, viewed: January 29, 2020 https://support.apple.com/en-au/HT204054

[15]. MetricKit documentation. Accessed: February 21, 2020: https://developer.apple.com/documentation/metrickit

___