CROSSMINER

Developer-Centric Knowledge Mining from Large Open-Source Software Repositories

Project Number 732223

D7.7 Developer Activity Monitoring - Final Version

Version 1.0
22 December 2018
Final

Public Distribution

FrontEndART

Project Partners: **Athens University of Economics & Business**, **Bitergia**, **Castalia Solutions**, **Centrum Wiskunde & Informatica**, **Eclipse Foundation Europe**, **Edge Hill University**, **FrontEndART**, **OW2**, **SOFTEAM**, **The Open Group**, **University of L'Aquila**, **University of York**, **Unparallel Innovation**

Project Partner Contact Information

Athens University of Economics & Business Diomidis Spinellis Patision 76 104-34 Athens Greece Tel: +30 210 820 3621 E-mail: dds@aueb.gr	**Bitergia** José Manrique Lopez de la Fuente Calle Navarra 5, 4D 28921 Alcorcón Madrid Spain Tel: +34 6 999 279 58 E-mail: jsmanrique@bitergia.com
Castalia Solutions Boris Baldassari 10 Rue de Penthièvre 75008 Paris France Tel: +33 6 48 03 82 89 E-mail: boris.baldassari@castalia.solutions	**Centrum Wiskunde & Informatica** Jurgen J. Vinju Science Park 123 1098 XG Amsterdam Netherlands Tel: +31 20 592 4102 E-mail: jurgen.vinju@cwi.nl
Eclipse Foundation Europe Philippe Krief Annastrasse 46 64673 Zwingenberg Germany Tel: +33 62 101 0681 E-mail: philippe.krief@eclipse.org	**Edge Hill University** Yannis Korkontzelos St Helens Road Ormskirk L39 4QP United Kingdom Tel: +44 1695 654393 E-mail: yannis.korkontzelos@edgehill.ac.uk
FrontEndART Rudolf Ferenc Zászló u. 3 I./5 H-6722 Szeged Hungary Tel: +36 62 319 372 E-mail: ferenc@frontendart.com	**OW2 Consortium** Cedric Thomas 114 Boulevard Haussmann 75008 Paris France Tel: +33 6 45 81 62 02 E-mail: cedric.thomas@ow2.org
SOFTEAM Alessandra Bagnato 21 Avenue Victor Hugo 75016 Paris France Tel: +33 1 30 12 16 60 E-mail: alessandra.bagnato@softeam.fr	**The Open Group** Scott Hansen Rond Point Schuman 6, 5th Floor 1040 Brussels Belgium Tel: +32 2 675 1136 E-mail: s.hansen@opengroup.org
University of L'Aquila Davide Di Ruscio Piazza Vincenzo Rivera 1 67100 L'Aquila Italy Tel: +39 0862 433735 E-mail: davide.diruscio@univaq.it	**University of York** Dimitris Kolovos Deramore Lane York YO10 5GH United Kingdom Tel: +44 1904 325167 E-mail: dimitris.kolovos@york.ac.uk
Unparallel Innovation Bruno Almeida Rua das Lendas Algarvias, Lote 123 8500-794 Portimão Portugal Tel: +351 282 485052 E-mail: bruno.almeida@unparallel.pt	

Table of Contents

Document Control

Version	Status	Date
0.5	Initial version	29 November 2018
0.8	Draft ready for internal review	3 December 2018
0.9	First pass corrections	19 December 2018
1.0	Final version	22 December 2018

Executive Summary

This document presents deliverable D7.7 (Developer Activity Monitoring - Final Version) of the CROSSMINER project. The deliverable is the final implementation of the Developer Activity Monitoring features, implemented in task T7.2 of WP7 as part of the CROSSMINER Eclipse IDE Plug-in. Developer activity monitoring collects information on how the developers use the IDE during their development activity in general, and how they use the features of the CROSSMINER Eclipse IDE Plug-in. Then it calculates process related metrics and sends them to the CROSSMINER server.

The deliverable covers 100% of the developer activity monitoring related technology and about 90% of the related use case requirements defined in deliverable D1.1 (Project Requirements).

1 Introduction

Figure 1 shows how the CROSSMINER Eclipse IDE Plug-in is connected to the CROSSMINER platform. The plug-in implements the Developer Activity Monitoring features and it is connected to the CROSSMINER server through the common CROSSMINER API. In this deliverable, we present the implementation of the Developer Activity Monitoring features in detail. These features enable the user to go through the whole process of measuring user activity: send the data to the server and use the captured metric values.

Section 2 gives an overview of the scenarios associated with the Developer Activity Monitoring features. We present the details of the implementation in Section 3. Section 5 links the current implementation status to the project requirements.

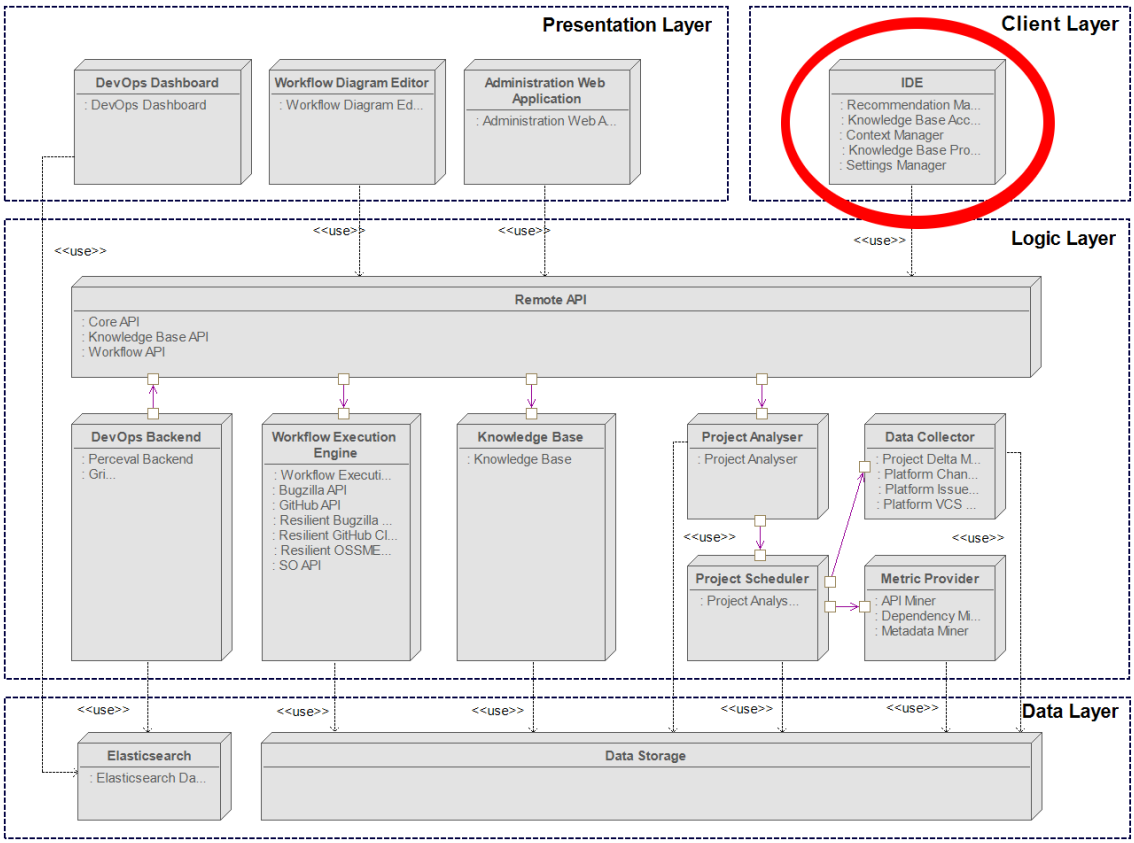

Figure 1: Location of the Integrated Development Environment in the CROSSMINER platform

2 Technical documentation

One of the main functionalities of the CROSSMINER Eclipse IDE Plug-in is to monitor and collect data on the developers' activity during the development process, then send the collected data to the CROSSMINER server for further processing. The server can calculate high-level metrics of the development process, and the Web-based Dashboard is able to show calculated metric values to developers. Users are able to authenticate themselves, after which they will be able to configure their own metrics set, and enable the server to compute the information they are interested in.

2.1 Scenarios

There are two general scenarios concerning the Developer Activity Monitoring logic implemented in the CROSSMINER Eclipse IDE Plug-in. These encapsulate the detection and use of the metrics. The first scenario shows how events and metric values are detected, calculated and stored. In the second scenario, the developer uses a specific model based on the captured low-level metrics to evaluate the system in development. Note, that in the second scenario the CROSSMINER Eclipse IDE Plug-in relies on the features of the Web-based Dashboard by means the plug-in provides the user with a link that points to the dashboard page which presents the metrics to the user. We use integration services, described in *D7.6: IDE Integration Services (Final Version)*, to send metrics related data to the CROSSMINER server.

2.1.1 Detection of Process Metrics

The main steps of this scenario are the recording of developer interactions (events), computing process related metrics, and sending them to the CROSSMINER server for further processing.

We show the components implementing these steps and their interaction in Figure 2. First, the CROSSMINER Eclipse IDE Plug-in collects events from Eclipse which are stored in a local database (cf. "1: monitor user activity" in Figure 2). Note, that the current implementation provides a graphical interface to the users where they can view and query the locally stored event chain. Please note that this GUI is only used for debugging purposes and will not be present in the final version of the CROSSMINER Eclipse IDE Plug-in. Then, the metrics are computed from the stored event chain (cf. "2: compute" in Figure 2). Finally, the metrics and some metadata are sent to the CROSSMINER server (cf. "3: send project id, ..." in Figure 2). The metadata includes a hashed user ID; it is used to enable the server to recognize if two sets of data originate from the same source (user) but without allowing the server to determine the exact identity of the user. A one-way cryptographic hash function will be used for this purpose that is unfeasible to invert.

2.1.2 Usage of Process Metrics

There are several ways to use process related metrics to guide developers (see an example in Figure 3). From the developer's point of view, the Web-based Dashboard has to be prepared by setting

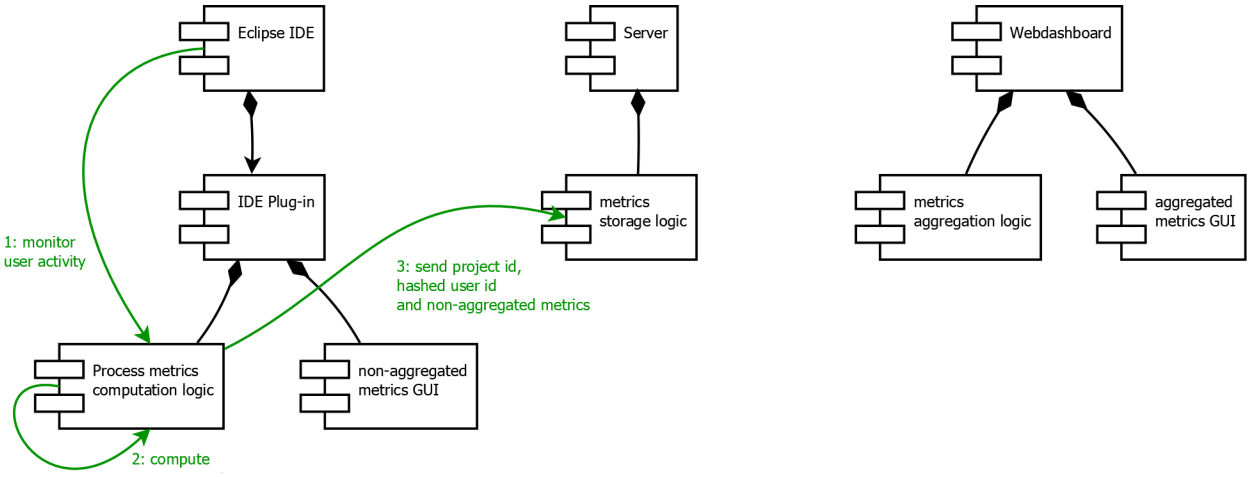

Figure 2: Overview of Detection Scenario for Using the CROSSMINER Eclipse IDE Plug-in

up a model and collecting necessary metrics. Next, the developer can query the evaluation of a component, such as a library. As the metrics-based evaluation is done by the Web-based Dashboard, the CROSSMINER Eclipse IDE Plug-in requests for the proper URL, and redirects the developer to the prepared page of the dashboard.

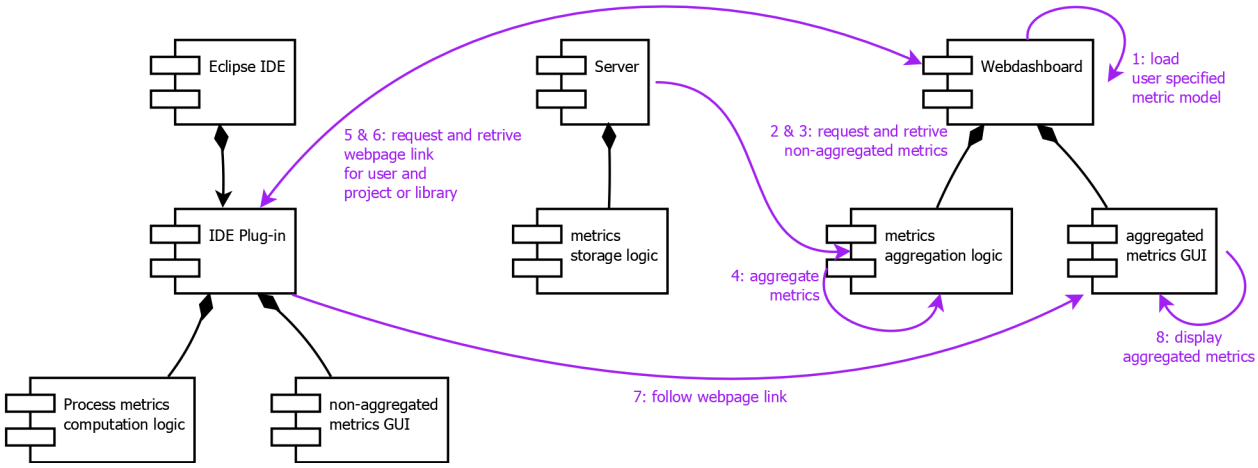

Figure 3: Overview of Usage Scenario for Using the CROSSMINER Eclipse IDE Plug-in

3 Implementation

The three main steps of the Developer Activity Monitoring process executed on the client side are shown in Figure 4. First, the CROSSMINER Eclipse IDE Plug-in collects several events from different components of Eclipse to cover all aspects of the development. Next, these events are stored in a local database. We chose OrientDB[1]; it is a graph-based database that fits into our storage and computation logic, eases the implementation of various metric computations, and has an appropriate license. Finally, the stored information is used to calculate low-level metrics, which will then be sent to the CROSSMINER Server.

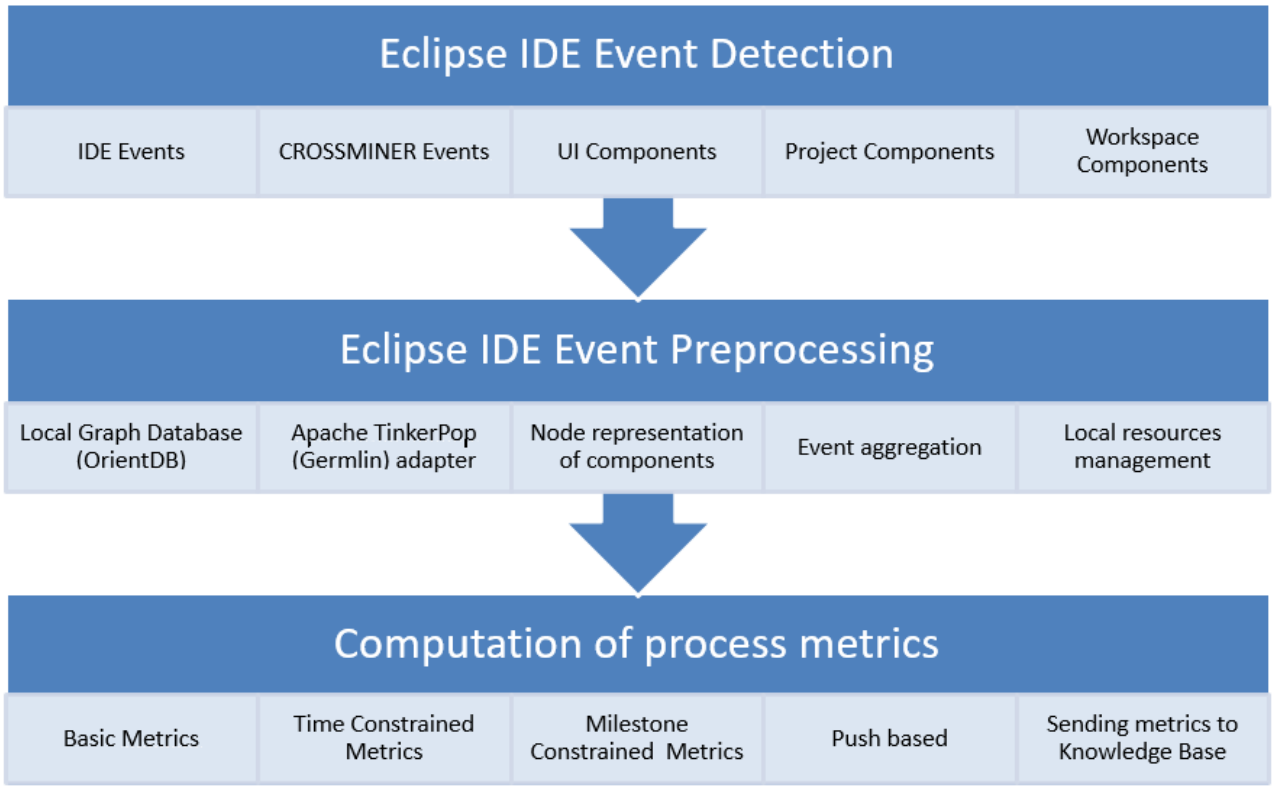

Figure 4: Developer Activity Monitoring Process and its components in the CROSSMINER Eclipse IDE Plug-in

3.1 Eclipse IDE Event Detection

The CROSSMINER Eclipse IDE Plug-in collects several types of events of the Eclipse IDE. One of the most important attributes of choosing the events was to observe the users' activities from different aspects. The events have been categorized; the resulting groups included IDE events, Plug-in related events, UI events, Project and Workspace related events.

[1] https://orientdb.com/

3.1.1 Types and Properties of Recorded Events

3.1.1.1 Document event The Document event type is based on the `org.eclipse.jface.text.DocumentEvent` class and used to detect all keypress events in the editor as well as to store affected files. The event is called after 10 button presses or after 10 ms, doing so we are able to reduce the size of the database. Properties of the event type includes *count, timestamp* and *type*, and it is connected to the affected file. This event type is useful, for example, to gather information on the user's typing frequency.

3.1.1.2 Project Structure Build event This is a special event that is in charge of mapping the hierarchy of the project. It creates the graph representation of the project in the database. It is called when we open Eclipse or open a project.

3.1.1.3 Idle event When there is no new event in the database for a pre-set amount of time we assume that to indicate the user is not using the IDE.

3.1.1.4 Part event The Part event type is based on the `org.eclipse.ui.IPartListener2` interface. It stores information about the life-cycle of given parts of the Eclipse application, e. g. activating, deactivating, and closing. The event has a *timestamp* and a *type* property, i. e. describing the behaviour of the given part, and is connected to the affected part. This event enables the reporting of Eclipse parts that are active or most frequently used.

3.1.1.5 Window event The Window event type is based on the `org.eclipse.ui.IWindowListener` interface. It stores information about the life-cycle of Eclipse windows in the same way as the **Part event** handles information about the parts. This event also has *timestamp* and *type* properties. It provides, for example, to check which parts of Eclipse application are active or most frequently used.

3.1.1.6 Eclipse close event This event type is based on the `org.eclipse.ui.IWorkbenchListener` interface. As this is the last event when the eclipse is shutting down, we use it as a milestone capturing events between two Eclipse application launches. It has a *timestamp* attribute.

3.1.1.7 Launch event The Launch event type is based on the `org.eclipse.debug.core.ILaunchListener` interface and stores information about code building and launching. The event has a *timestamp* and contains information about the *type* of launch mode. For example, the event captures whether a build was started in debugging, test or normal mode.

3.1.1.8 Resource element event The Resource element event type is based on the `org.eclipse.ui.texteditor.IElementStateListener` interface. We use it to capture actions on files, such as save and delete. The event has a *timestamp* and a *type* property and connects to the affected file. Furthermore, we plan to use this event type to collect information on the refactoring process.

3.1.1.9 Class path event The Class path event type is based on the `org.eclipse.jdt.core.IElementChangedListener` interface. Classpath additions and removals can be captured by this type of event. The event has *timestamp* and *type* properties. This event type can be used to check the ratio between manual and CROSSMINER Eclipse IDE Plug-in indicated library changes.

3.1.1.10 CROSSMINER events The CROSSMINER events are generated when CROSSMINER Eclipse IDE Plug-in features used and are planned to be used to calculate plug-in feature usage metrics.

3.1.2 Categorization of Event Related Components

Event detection requires additional, non-event resources to be stored in connection to the events. These resources are important for the grouping of events and contain extra information for metric calculation. This additional information enables us to examine events in different scales of the project.

3.1.2.1 File Abstract representation of the Eclipse's `IFile` resources (e. g. Text Editor input). It helps the CROSSMINER Eclipse IDE Plug-in to capture files affected by events and, in later stages of the development, enables the calculation of metrics associated with file usage in the project. This resource stores the *name* of the File.

3.1.2.2 Part The Parts represent different parts of the Eclipse application. It enables the CROSSMINER Eclipse IDE Plug-in to track the most active parts of the Eclipse application and represents the basis for grouping different types of events. Is stores the *title* of the Part.

3.1.2.3 Window The windows represent the parent Eclipse application window, and it can also be used to group items.

3.1.2.4 Package The Package will represent the Package, and will be connected to the **File**, **Package** or **Project** items representing the package structure.

3.1.2.5 Project The projects will represent the projects, and will be connected to the **File** or **Package** items representing the representing the project structure.

3.2 Eclipse IDE Event Preprocessing

The events and items are represented in a graph database as a graph. Every single event and resource is a vertex in this graph. The edges of the graph represent relations between the connected events and items.

3.2.1 Representation and Event Storing Logic

In the CROSSMINER Eclipse IDE Plug-in, the event handling system is based on the `EventManager` class; all event listeners call this class to process the given event. The `EventManager` is responsible to send events to the `Gremlin Adapter`. The `EventManager` can also filter event types before their submission to the database, and thus reduce database workload.

All of the collected events have their own event classes which are inherited from the `Event` class. Every event class stores information about the event and implements a `toNode()` method which describes the way it can be inserted into the database. All of these event classes are processed by the `EventManager`, which sends them to the `Gremlin Adapter`. Moreover the `EventManager` class is responsible for the subscription to the selected set of event listeners.

The `DatabaseManager` class implements the client database connection. The actual implementation uses the OrientDB[1] database. As we are using Gremlin for communication with the DBMS, the actual database system can be replaced by different graph database given that it implements the TinkerPop™[2] interface. The `DatabaseManager` is responsible for creating the database connection and communicate with the Gremlin Adapter. Although all event class objects can connect themselves to their local resources, the `Gremlin Adapter` will join individual events into an event chain. Every event is connected to the next event with a `Next` edge. Events and resources are connected by `SubjectResource` edges and thus facilitate the filtering or retrieval of resource-related events. Resources can also be connected to each other; for example, `File` and `Project` nodes are connected to represent the project structure, facilitate its exploration, detect of hot-spot classes, and calculate metrics. An example graph representation of an event chain is presented in Figure 5. The dark gray nodes are events and the light gray nodes denote resources. The individual subgraphs consisting of light gray nodes are different projects.

In the following paragraph, we will elaborate on each node and their connection in the underlying graph database. We use the previously introduced categorization of nodes (events and components) to define the scheme of these.

3.2.1.1 Nodes and Their Properties In the database structure, we use two different labels to distinguish nodes. There are `Event` and `Resource` labels and all of the nodes have a `VertexType`

[2] `http://tinkerpop.apache.org/`

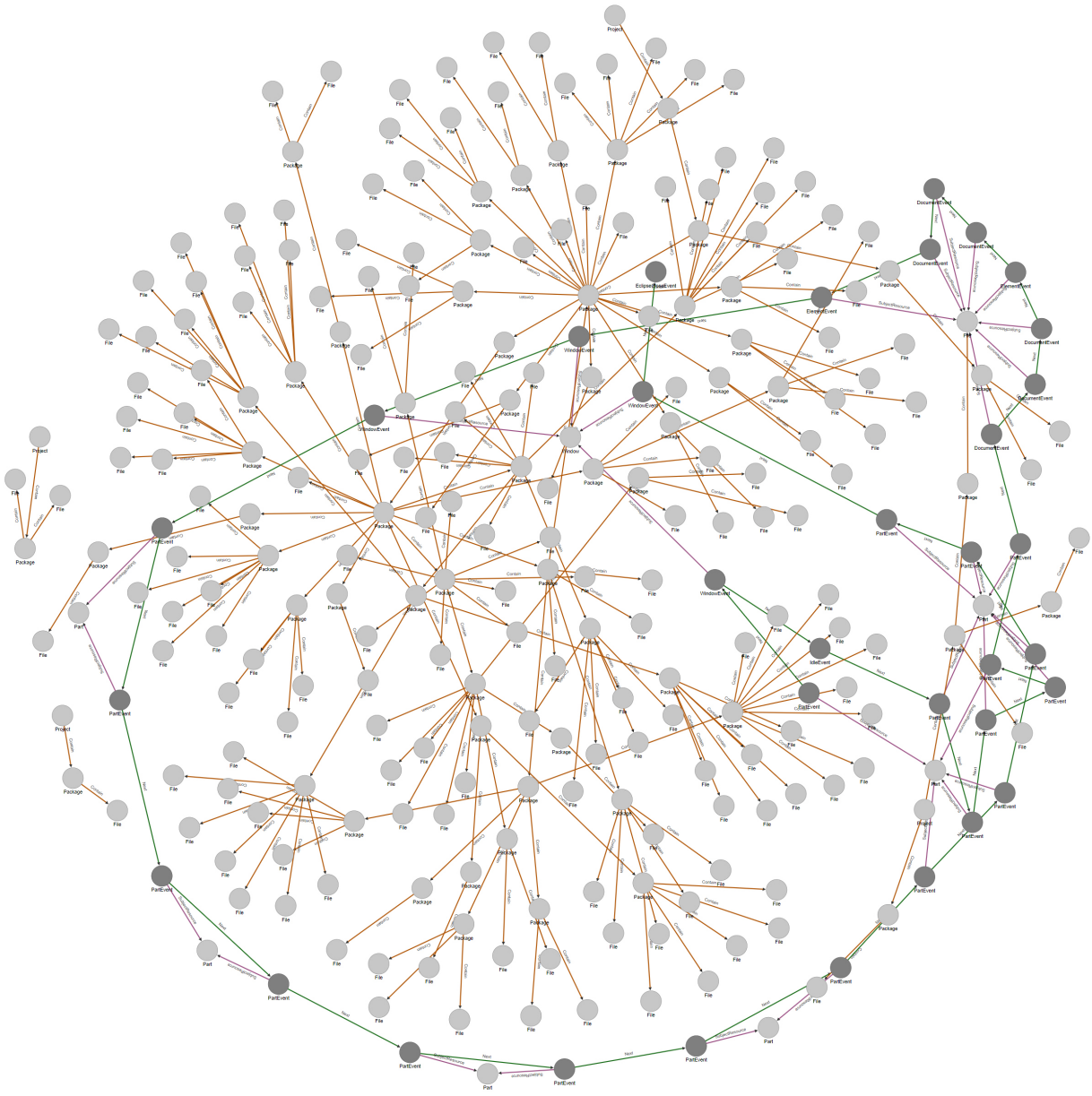

Figure 5: An example possible graph in our database structure.

property which describes the current event or resource type. We can also group the events based on whether they are project related or not. For example, a `DocumentEvent` is connected to a file in the project, so the event is linked to the project while, a `Eclipse Close Event` cannot be connected to any of the projects. The events which are related to a project are connected to it through a `Related Project` edge.

3.2.1.1.1 Abstract Event Node All event types are inherited from this class. Event properties are listed in Table 1. There is a `Timestamp` property in this class what the other event classes inherit. We use Timestamp for Time-based metrics and we delete those nodes that are too old.

Name	Java type	OrientDB type	Range
timestamp	Date	DateTime	representing the time of the event

Table 1: Properties of Event nodes

3.2.1.1.2 Document Event Node Document Event properties are listed in Table 2. As we mentioned in the previous paragraph, every event node has a `VertexType` property which describes the concrete event type. We use event aggregation for this event type because this is the most frequent event and we considerably reduce the database size after buffering it. It stores how many key presses happened before the last event insertion of the same type.

Name	Java type	OrientDB type	Range
Type	String	String	documentChanges
Count	String	String	1-10
VertexType	String	String	DocumentEvent

Table 2: Properties of Document Event node

3.2.1.1.3 Classpath Change Event Node Classpath Change Event properties are listed in Table 3. There are four types of Classpath Change Event, which is stored in `Type` property. We distinguish manual and CROSSMINER indicated classpath changes for future metric calculation.

Name	Java type	OrientDB type	Range
Type	String	String	ADDED, DELETED, CROSSMINER _ADDED, CROSSMINER _DELETED
VertexType	String	String	ClasspathChangeEvent

Table 3: Properties of Classpath Change Event node

3.2.1.1.4 Eclipse Close Event Node Eclipse Close Event properties are listed in Table 4. This event is used to detect the closing of eclipse, it has no special properties.

Name	Java type	OrientDB type	Range
VertexType	String	String	EclipseCloseEvent

Table 4: Properties of Eclipse close Event node

Name	Java type	OrientDB type	Range
VertexType	String	String	IdleEvent

Table 5: Properties of Idle Event node

3.2.1.1.5 Idle Event Node Idle Event properties are listed in Table 5. This event, just like Eclipse Close Event, has no special properties, we use it only to detect when the user is idle.

3.2.1.1.6 Launch Event Node Launch Event properties are listed in Table 6. There are three types of Launch Event: we distinguish run, debug and test launches for future metric calculation. The type is stored in the `Type` property.

Name	Java type	OrientDB type	Range
Type	String	String	run, debug, test
VertexType	String	String	LaunchEvent

Table 6: Properties of Launch Event node

3.2.1.1.7 Part Event Node Part Event properties are listed in Table 7. Part Event nodes contain information about the part action. For example, if we open a file in Eclipse, text editor part is opened, brought to the top, activated and its visibility is changed.

Name	Java type	OrientDB type	Range
Type	String	String	ACTIVATED, BROUGHT_TO_TOP, CLOSED, DEACTIVATED,OPENED, HIDDEN, VISIBLE, INPUT_CHANGED
VertexType	String	String	PartEvent

Table 7: Properties of Part Event node

3.2.1.1.8 Resource Element Event Node Resource Element Event properties are listed in Table 8. The `Type` property contains information whether the resource is saved or deleted.

Name	Java type	OrientDB type	Range
Type	String	String	SAVED, DELETED
VertexType	String	String	ElementEvent

Table 8: Properties of Resource Element Event node

3.2.1.1.9 Window Event Node Window Event properties are listed in Table 9. It stores information about the life-cycle of Eclipse windows in the same way as the **Part event** handles information about the parts.

Name	Java type	OrientDB type	Range
Type	String	String	ACTIVATED, CLOSED, DEACTIVATED, OPENED
VertexType	String	String	PartEvent
Title	String	String	Workspace

Table 9: Properties of Window Event node

3.2.1.1.10 CROSSMINER Search Usage Event Node CROSSMINER Search Usage Event properties are listed in Table 10. This event is used for plug-in related metric computation.

Name	Java type	OrientDB type	Range
VertexType	String	String	CrossminerSearchUsageEvent

Table 10: Properties of CROSSMINER Search Usage Event node

3.2.1.1.11 CROSSMINER Search Success Event Node CROSSMINER Search Success Event properties are listed in Table 11. This event is used for plug-in related metric computation.

Name	Java type	OrientDB type	Range
VertexType	String	String	CrossminerSearchSuccesEvent

Table 11: Properties of CROSSMINER Search Success Event node

3.2.1.1.12 CROSSMINER Library Usage Event Node CROSSMINER Library Usage Event Node properties are listed in Table 12. This event is used for plug-in related metric computation.

Name	Java type	OrientDB type	Range
VertexType	String	String	CrossminerLibraryUsageEvent

Table 12: Properties of CROSSMINER Library Usage Event Node node

3.2.1.2 Edges and Their Properties All events are connected to the following event with a `Next` edge. Those events which are related to Resources have a `Subject Resource` edge to their resources. There are also connection among the resources, for example, project hierarchy represented with `Contains` edges. A concrete example is shown on Figure 6. As we can see, events are connected with `Next` edges, the `Document Events` node have a `Subject Resource` edge to the file which is affected. The `Package` node `Contains` three `File` nodes.

3.2.1.3 Representing the Chain of Events We used the above-detailed nodes and edges to represent sequential events during the development of the target application. The event chain contains all user indicated events in chronological order (Figure 7).

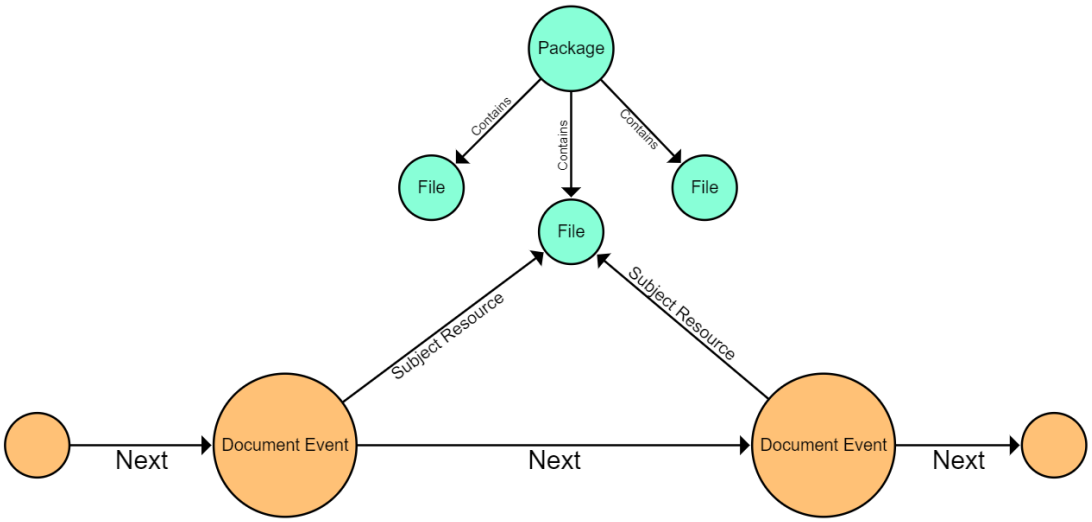

Figure 6: An example of the Document Event node

3.2.1.4 Representing the Project Hierarchy To capture Resource related events, we need to represent the project, its components, ant their hierarchical connections.

After an Eclipse launch or Project open, the plug-in automatically detects the project structure and stores it in the local database. These representations are static and the events which have a related file or project resource are connected to these nodes. To identify a project we use a `.CROSSMINER` helper file, that is stored in the project folder. We travel recursively through the project tree, starting from the project by using depth-first search to examine relations between project resources. We use this traversal method to identify the resources for resource-related events too. If we find a non-represented package or file during the traversal, the algorithm will simply add it to the project hierarchy representation. By using this method we are able to explore those files which are added from outside of Eclipse. This type of hierarchy is shown on Figure 8.

3.2.2 Local Resource Management and Further Optimization

As described above, events, connected resources, their attributes, and relations are stored in a local database on the plug-in (client) side. Due to the frequency and number of events that are generated by the CROSSMINER Eclipse IDE Plug-in, the size of this database increases very fast. Note, that the available storage space is limited and the large size of the database hinders fast metric calculation. So, it is important to manage the database size by deleting events or other entities.

If the users decide to delete the local database content, they are able to do it manually any time by choosing the related option in the IDE. Another option is based on constraints the age of events. Thus, if an event gets older than a preset time limit (determined by the user) or the widest time-window which is calculated, it is deleted from the database. This solution allows the database size to

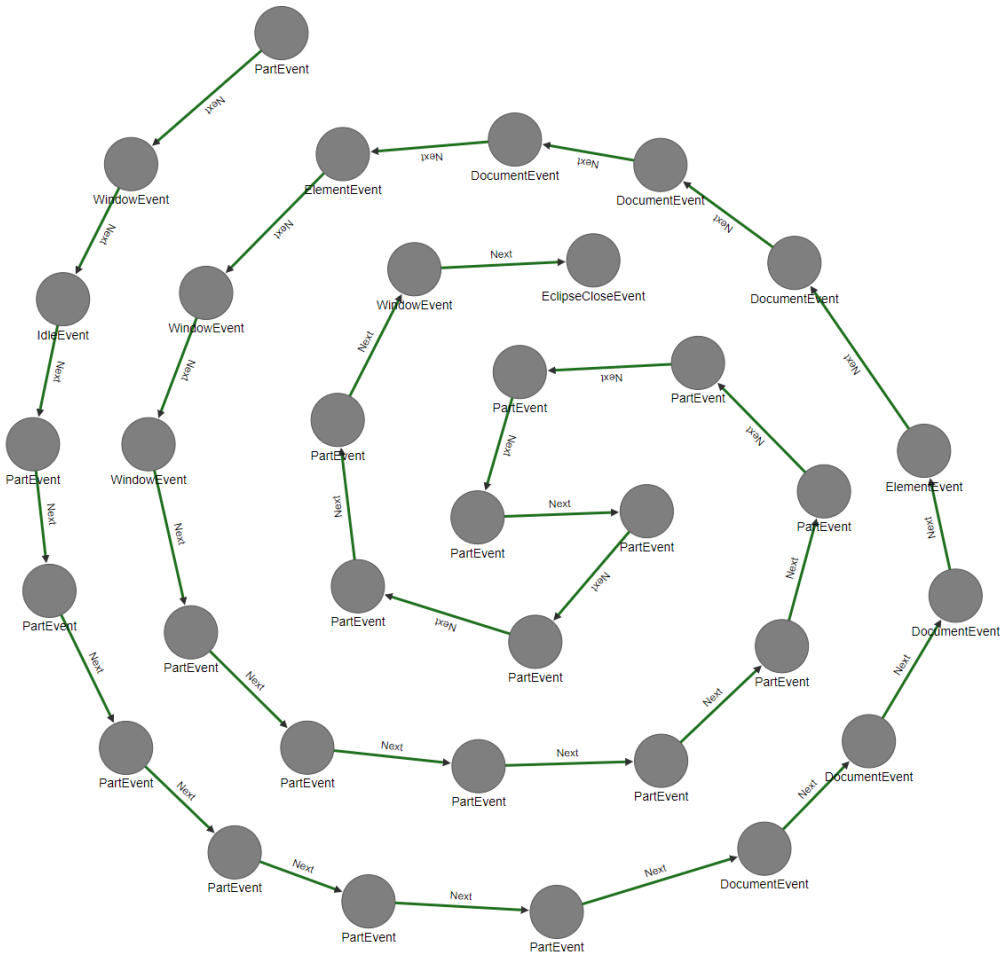

Figure 7: An example of the Event chain

change dynamically, depending on the frequency of the events. Finally, when the developers disable unwanted metrics, we suppress the collection of unnecessary events.

Besides the events, the database also stores related resources. As the resources are static items, i. e. non-temporal events, they cannot be simply deleted from the database even if they are not connected to other events at the given time. On the other hand, being static, the storage space these items consume will not grow as dynamically as in the case of events. This allows us to ignore the dynamic management of these resources.

3.3 Computation of Process Metrics

Events are collected for the purpose to compute process metrics. The values of these metrics are used to describe the development from different aspects. For example, we can calculate the average save

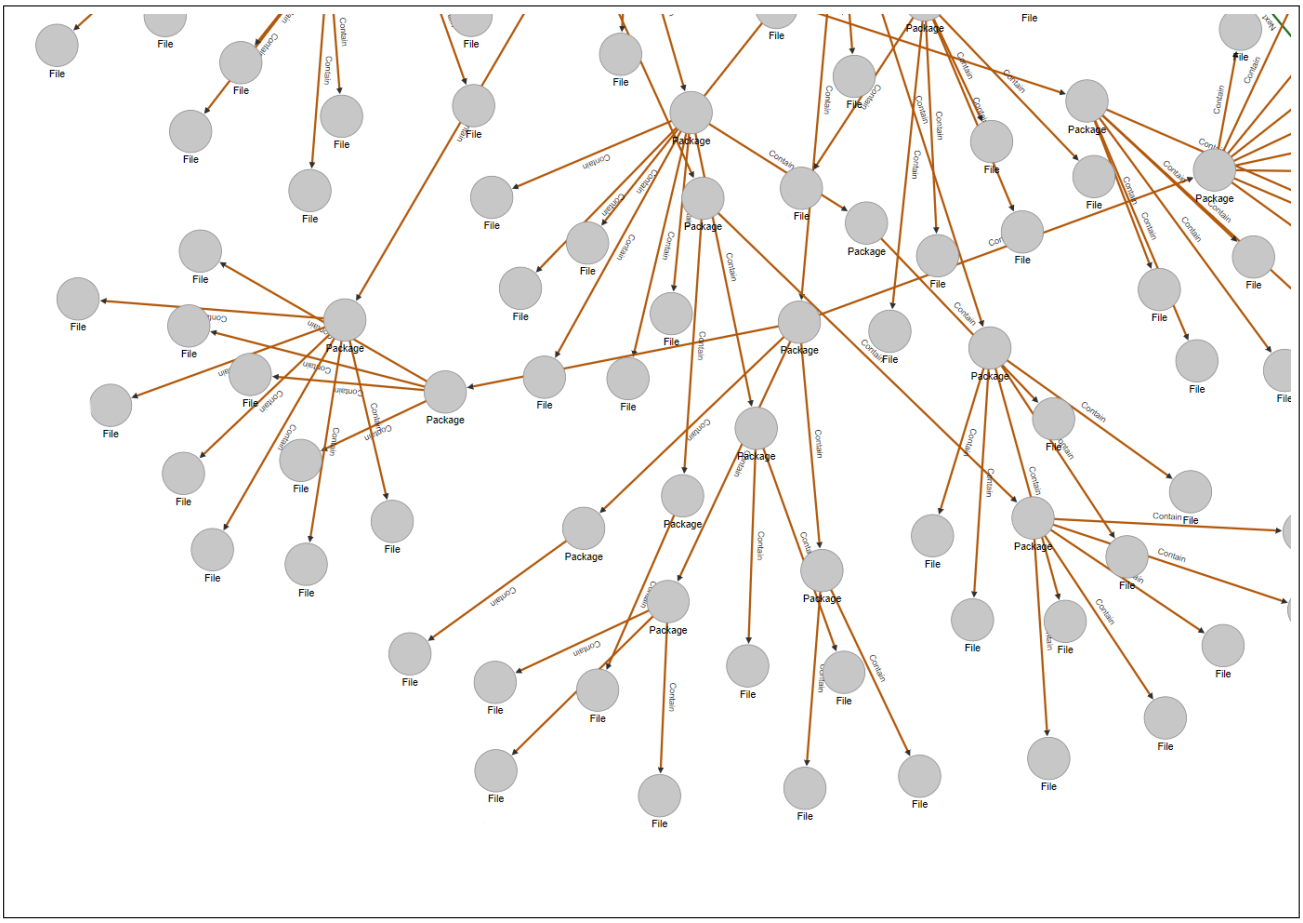

Figure 8: Project hierarchy subtree from our database structure.

rate in a dedicated time period or count different development-related interactions. We call these *basic metrics*. In the following subsections, we will introduce each of these.

3.3.1 Basic Metrics

3.3.1.1 File-Access-Rate Basic Metric Intended to express the Average count of Java source files open and brought to the top.

We use the following events and components to calculate this metric.

- Part event
- File resource

3.3.1.2 Working-time Basic Metric Used to express the amount of time while the user works on different Java files.

We use the following events and components to calculate this metric.

- Document event

- Idle event
- File resource

3.3.1.3 Testing-rate Basic Metric

Used to express the amount of test execution by the user.

We use the following events and components to calculate this metric.

- Launch event
- File resource

3.3.1.4 GUI-usage-rate Basic Metric

Used to express the ratio of graphical interface interaction by the user.

We use the following events and components to calculate this metric.

- Part event
- Window resource

3.3.1.5 Modification-rate Basic Metric

Used to express the ratio of file modifications by the user.

We use the following events and components to calculate this metric.

- Document event
- File resource

3.3.1.6 CROSSMINER-Search-Success Basic Metric

Used to express the amount of successful CROSSMINER Eclipse IDE Plug-in search by the user.

We use the following events and components to calculate this metric.

- CROSSMINER search success event

3.3.1.7 CROSSMINER-Search-Usage Basic Metric

Used to express the amount of using the CROSSMINER Eclipse IDE Plug-in search function by the user.

We use the following events and components to calculate this metric.

- CROSSMINER search usage event

3.3.1.8 CROSSMINER-Library-Usage Basic Metric

Used to express the amount of using the CROSSMINER Eclipse IDE Plug-in library search function by the user.

We use the following events and components to calculate this metric.

- CROSSMINER library usage event

3.3.2 Aggregation strategies

To transform the event chain into metric values, we use two similar strategies. Both strategies process subsequent events and compute metric values based on the number or the property values of certain events. The two strategies differ in how the start and end events of the sequence are determined.

The first, so-called *time-based strategy* uses fixed time windows to compute the metrics. In this strategy, we split the day into non-overlapping equal-length time periods (called *window*s), and compute the metric values using the events in a single window. Metrics are computed for all windows. The computation of this kind of metrics are illustrated on Figure 9 by the purple lines above the event chain.

Careful selection of the time windows is required. Our experiences have shown that time window must be distinct and fixed within a day. But the non-overlapping feature does not mean that we cannot have different window settings at the same time. We can define non-overlapping 1-hour long windows for some metrics while using 2-hours long windows to compute other metrics for the same event chain (as shown in Figure 9 by the solid and dashed purple lines). For example, we can create a metric called "Manual and CROSSMINER library change ratio in an hour" and similar metrics with two and four hours long windows.

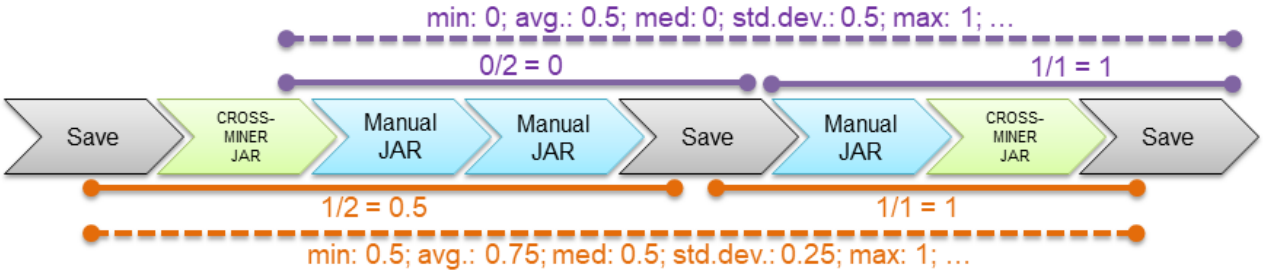

Figure 9: Time and milestone based metric computation

We call the second strategy *milestone-based strategy*. In this case, instead of the time-based windows, we use specific events to split and limit the event chains on which the metric values are calculated. In Figure 9, the orange lines illustrate the milestone-based strategy, where the save events have been chosen as milestones.

3.3.2.1 Aggregation Functions There are several aggregation functions to be used with either of the previously described strategies. In this section, we elaborate the well known statistical function we have chosen to implement.

Average The common arithmetic mean.

Summation The sum of all items.

Standard deviation Used to quantify the amount of variation or dispersion of the set of values.

As stated earlier the calculation of basic metrics are constrained with various time-windows. In this section, we list both the milestone and the duration-based time window definitions included in the CROSSMINER Eclipse IDE Plug-in.

3.3.2.1.1 Duration-based windows Time-based windows are defined with a duration and a count of inspected windows counted from the present. Duration defines the size of the window and the count describes how many of these windows will be analyzed (as shown in Table 13). For example, a short-term time window works with 24 1-hour windows. These duration and count values are based on our experiences and common practices of the developers.

Name	Duration (hours)	Count
short-term	1	24
mid-term	24	7
long-term	168	4
overview-term	24	28

Table 13: Time-based windows which is used for metric aggregation.

3.3.2.1.2 Milestone-based windows The milestone-based windows are calculated from an event and a count. The event describes the event type what we are using for splitting the event chain to get the windows. The count just like before describes how many of these windows will be inspected. An example is shown in Table 14. In this, a session window uses Eclipse close events to split the chain, and using 15 of these windows are used. These event and count values are also from our daily observation.

Name	Event	Count
session	Eclipse close	15
engagement	Save	15
last-session	Eclipse close	1
last-engagement	Save	1
coding-session	Launch	15
last-coding-session	Launch	1

Table 14: Milestone-based windows which is used for metric aggregation

3.3.3 Calculated Metrics

As mentioned above, the current implementation uses the OrientDB database which is accessed using the Gremlin graph database query language[3] provided by Apache TinkerPop™. Using Gremlin, we are able to filter events by their type or other properties, and we can follow the edges to find the

[3] `http://tinkerpop.apache.org/gremlin.html`

connections between the nodes. Gremlin provides us various tools to manage our graphs. The basis of all of our metrics is a gremlin query which lists or counts the dedicated vertices. We use various aggregations on this data set to calculate the defined metrics. We have implemented some aggregation methods, but the list can be easily expanded if needed.

The following example selects and lists all the vertices that are save events (`VertexType.ELEMENT_EVENT` and `ResourceElementStateType.SAVED`) and contained in a time window (between `begin` and `end`).

```
List<Vertex> list = graphTraversalSource.V()
.has("VertexType", VertexType.ELEMENT_EVENT)
.has("Type",ResourceElementStateType.SAVED)
.where(__.has("TimeStamp", P.gte(begin)))
.where(__.has("TimeStamp", P.lte(end)))
.order().by("TimeStamp",Order.incr)
.toList();
```

Listing 1: Simple Gremlin query, to select events in a time window.

3.3.3.1 Calculated Metrics Any calculated metric consists of a basic-metric and an aggregation (which include the strategy and the function) component. In the following sections we list all the calculated metrics that are built from a basic metric, a time window and an aggregation function. The tables for each metric show which metric windows and which aggregation function is used for the calculations.

	average	**stdev**	**sum**
short-term	✓		
mid-term	✓		
long-term	✓		
overview-term	✓	✓	
session			
engagement			
last-session			
last-engagement			
coding-session			✓
last-coding-session			

Table 15: Different calculation methods for CROSSMINER-library-usage metric

3.3.3.1.1 CROSSMINER-library-usage The purpose of this metric is to analyze the plug-in provided library-related features. The different methods to calculate this metric are shown in Table 15.

	average	stdev	sum
short-term	✓		
mid-term	✓		
long-term	✓		
overview-term	✓	✓	
session			
engagement			
last-session			
last-engagement			
coding-session			✓
last-coding-session			

Table 16: Different calculation methods for CROSSMINER-search-usage metric

3.3.3.1.2 CROSSMINER-search-usage This metric is to analyze the plug-in provided search function. The different methods to calculate this metric are shown in Table 16.

	average	stdev	sum
short-term	✓		
mid-term	✓		
long-term	✓		
overview-term	✓	✓	
session			
engagement			
last-session			
last-engagement			
coding-session			✓
last-coding-session			

Table 17: Different calculation methods for CROSSMINER-search-success metric

3.3.3.1.3 CROSSMINER-search-success This metric helps to analyze the success of the plug-in provided library search function. The different methods to calculate this metric are shown in Table 17.

3.3.3.1.4 Modification-rate Used to express the rate of file modification by the user. The different methods to calculate this metric are shown in Table 18.

	average	stdev	sum
short-term	✓		
mid-term	✓		
long-term	✓		
overview-term	✓	✓	
session			
engagement			
last-session			
last-engagement			
coding-session			
last-coding-session			

Table 18: Different calculation methods for Modification-rate metric

	average	stdev	sum
short-term	✓		
mid-term	✓		
long-term	✓		
overview-term	✓	✓	
session			
engagement			
last-session			
last-engagement			
coding-session			
last-coding-session			

Table 19: Different calculation methods for GUI-usage-rate metric

3.3.3.1.5 GUI-usage-rate Used to express the rate of graphical interface interactions by the user. The different methods to calculate this metric are shown in Table 19.

3.3.3.1.6 Trust This metric expresses the user's trust towards the IDE. It is calculated by measuring the rate of file modification between saves, i. e. how much changes are kept unsaved. The different calculation methods to calculate this metric are shown in Table 20.

3.3.3.1.7 Confidence This metric covers a similar concept like trust, but it expresses the self-confidence of the developers. It captures the amount of file modification without execution of the system under development. The different methods to calculate this metric are shown in Table 21.

	average	stdev	sum
short-term			
mid-term			
long-term			
overview-term			
session			
engagement	✓	✓	
last-session			
last-engagement	✓		
coding-session			
last-coding-session			

Table 20: Different calculation methods for Trust metric

	average	stdev	sum
short-term			
mid-term			
long-term			
overview-term			
session			
engagement			
last-session			
last-engagement			
coding-session	✓	✓	
last-coding-session	✓		

Table 21: Different calculation methods for Confidence metric.

3.3.3.1.8 Testing-rate It is used to express the amount of test execution by the user. The different calculation methods of this metric are shown in Table 22.

3.3.3.1.9 Working-time We use it to express the amount of time while the user works on different Java files. The different ways to calculate this metric are shown in Table 23.

3.3.3.1.10 File-access-rate Intend to express the Average count of Java source file opens and focuses. The different methods to calculate this metric are shown in Table 24.

	average	stdev	sum
short-term	✓		
mid-term	✓		
long-term	✓		
overview-term	✓	✓	
session			
engagement			
last-session			
last-engagement			
coding-session	✓		
last-coding-session			

Table 22: Different calculation methods for Testing-rate metric

	average	stdev	sum
short-term			✓
mid-term			✓
long-term			✓
overview-term			✓
session			
engagement			
last-session			
last-engagement			
coding-session			✓
last-coding-session			

Table 23: Different calculation methods for Working-time metric

3.3.4 Sending metrics to the CROSSMINER server

The metric values are sent to the server in JSON format. This data contains a user identifier field (which could be used to authenticate the user without revealing its real identity) and a list of projects. The projects are identified with their GitHub URL. They contain a list of metrics which are computed from project-related events. Each metric has an identifier and a value property. The value of a metric is always represented as a double precision floating point number. For the work-time related metrics, we also includ a helper property, namely *fully qualified name*, which will contain the path to the subject Java source file. This implementation contains all of the information required by the server in order to process the metric object.

The CROSSMINER Eclipse IDE Plug-in sends metrics to the server in every predefined time period using a push-based communication method. The user is able to configure the time period at the client

	average	stdev	sum
short-term	✓		
mid-term	✓		
long-term	✓		
overview-term	✓	✓	
session			
engagement			
last-session			
last-engagement			
coding-session	✓		
last-coding-session			

Table 24: Different calculation methods for File-access-rate metric.

side. Moreover, users have to set their CROSSMINER authentication key in the CROSSMINER Eclipse IDE Plug-in settings to use this feature; without authentication, the server can't identify which client is sending the metrics. Note, that authentication is used only to differentiate between individual users and not to connect the measured metric values to specific users.

```
{
"projects": {
"https://github.com/crossminer/crossminer/tree/dev/eclipse-based-ide": {
"modification_rate": {
"shortterm_modificaton_rate": 121241,
"midterm_modificaton_rate": 12412,
"longterm_modificaton_rate": 12312,
"overview_modificaton_rate": 12312,
"divergency_modificaton_rate": 12312
},
"gui_usage_rate": {
"shortterm_gui_usage_rate": 121241,
"midterm_gui_usage_rate": 12412,
"longterm_gui_usage_rate": 12312,
"overview_gui_usage_rate": 12312,
"divergency_gui_usage_rate": 12312
},
"working-time:CROSSMINER/Usermonitoring/Event/EventManager.java": {
"shortterm_working-time": 121241,
"midterm_working-time": 12412,
"longterm_working-time": 12312,
```

```
"overview_working-time": 12312,
"divergency_working-time": 12312,
"per_coding_session_working-time": 12312,
"fully_qualified_name": "CROSSMINER/Usermonitoring/Event/
  EventManager.java"
},
"working-time:CROSSMINER/Usermonitoring/Gremlin/GremlinAdapter.
  java": {
"shortterm_working-time": 121241,
"midterm_working-time": 12412,
"longterm_working-time": 12312,
"overview_working-time": 12312,
"divergency_working-time": 12312,
"per_coding_session_working-time": 12312,
"fully_qualified_name": "CROSSMINER/Usermonitoring/Gremlin/
  GremlinAdapter.java"
},
"working-time:CROSSMINER/Usermonitoring/Vertices/VertexProperty.
  java": {
"shortterm_working-time": 121241,
"midterm_working-time": 12412,
"longterm_working-time": 12312,
"overview_working-time": 12312,
"divergency_working-time": 12312,
"per_coding_session_working-time": "12312",
"fully_qualified_name": "CROSSMINER/Usermonitoring/Vertices/
  VertexProperty.java"
}
},
"https://github.com/yomotsu/camera-controls": {
"modification_rate": {
"shortterm_modificaton_rate": 121241,
"midterm_modificaton_rate": 12412,
"longterm_modificaton_rate": 12312,
"overview_modificaton_rate": 12312,
"divergency_modificaton_rate": 12312
},
"gui_usage_rate": {
"shortterm_gui_usage_rate": 121241,
"midterm_gui_usage_rate": 12412,
"longterm_gui_usage_rate": 12312,
"overview_gui_usage_rate": 12312,
```

```
"divergency_gui_usage_rate": 12312
},
"working-time:Camera/Control/Camera/CameraControl.java": {
"shortterm_working-time": 121241,
"midterm_working-time": 12412,
"longterm_working-time": 12312,
"overview_working-time": 12312,
"divergency_working-time": 12312,
"per_coding_session_working-time": 12312,
"fully_qualified_name": "Camera/Control/Camera/CameraControl.java"
},
"working-time:Camera/Control/Camera/Axis.java": {
"shortterm_working-time": 121241,
"midterm_working-time": 12412,
"longterm_working-time": 12312,
"overview_working-time": 12312,
"divergency_working-time": 12312,
"per_coding_session_working-time": 12312,
"fully_qualified_name": "Camera/Control/Camera/Axis.java"
},
"working-time:Camera/Resources/Shaders/Shader.java": {
"shortterm_working-time": 121241,
"midterm_working-time": 12412,
"longterm_working-time": 12312,
"overview_working-time": 12312,
"divergency_working-time": 12312,
"per_coding_session_working-time": 12312,
"fully_qualified_name": "Camera/Resources/Shaders/Shader.java"
}
}
}
}
```


4 Developer Activity Monitoring Control

The user has full control over the Developer Activity monitoring. In this section, we elaborate on the various Developer Activity Monitoring related settings and customization functions.

4.1 Documentation

The developer is able to examine the collected metrics and the related event using the integrated preferences page. This UI also displays a short description for each of these with a small example to illustrate its usage. The user could find further details about these in the included user manual, which is also presented in a PDF format for convenience.

4.2 Changing Set of Detected Metrics

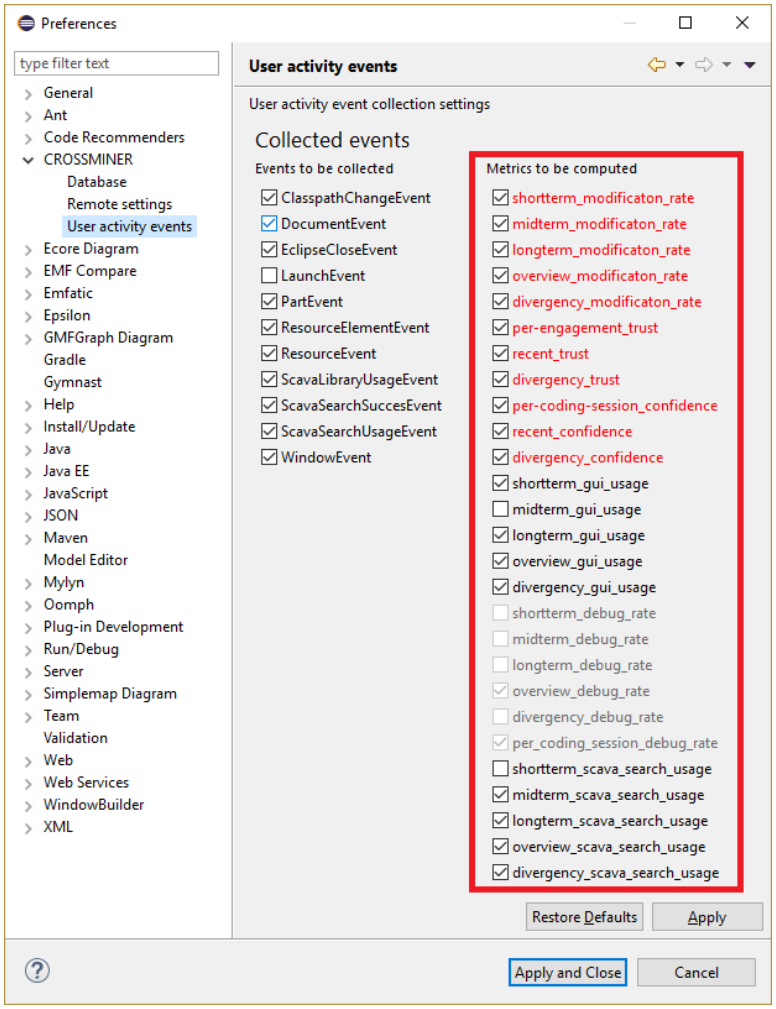

Figure 10: Filtering detected metrics

In the preferences page, the user can disable the computation of various metrics, or whole monitoring process. The user is able to delete the local database if it is desired.

The deactivation of metrics does not change the collection of underlying events since there are usually more than one metrics use them. To help the user to make the proper customization, we highlighted

the related events for the selected metric. These settings are shown on the right side of Figure 10. If an event type is unnecessary because none of metric uses it, then it will not be stored in the database.

4.3 Enable and Disable Event's Collection

It is possible to activate or disable the collection for each event individually. After an event type is disabled those metrics which rely on that event type are not calculated (Figure 11).

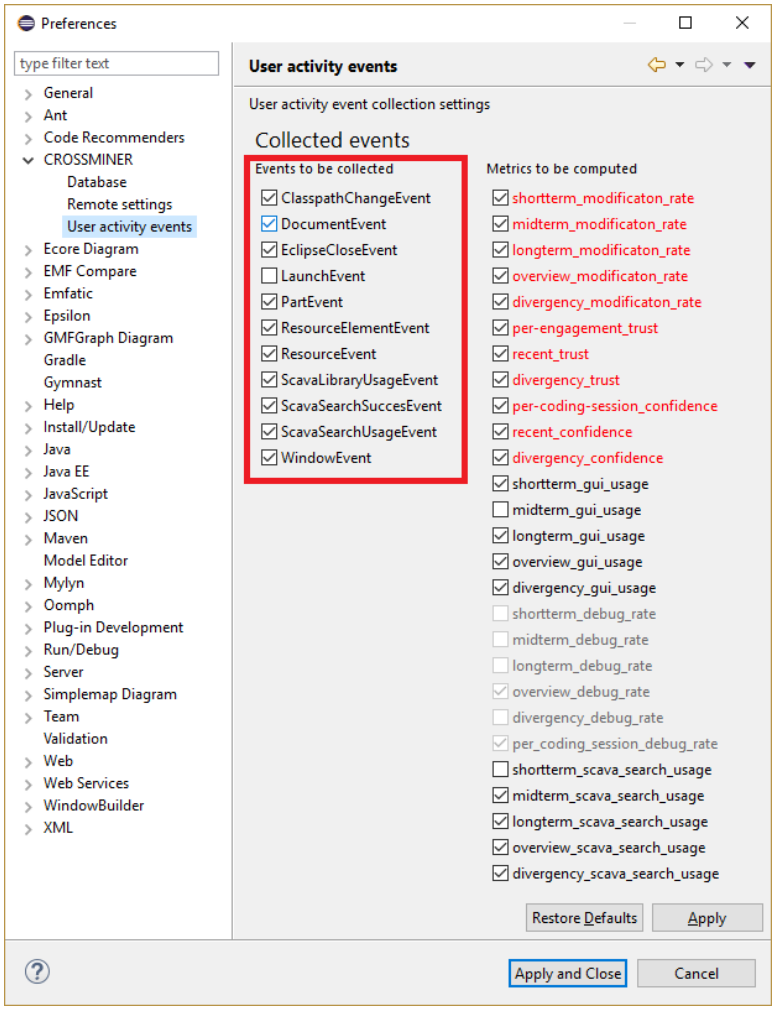

Figure 11: Filtering collected events

4.4 Parameterized Events

There is type of events that requires custom parameters which are usually different for each user. For example, for the Idle event the user is able the specify the duration of an event-less period which identified as idle time. These properties have default values based on our observations.

Idle event The duration of event-less period.

Document Change event Count of subsequent keystrokes or the longest delay between them.

5 Conclusion

In this deliverable the CROSSMINER Eclipse IDE Plug-in was extended with Developer Activity Monitoring features. In this section, first we overview the implemented functionality (strictly concentrating on the Developer Activity Monitoring related ones).

In the following subsections we show how the technical (Section 5.1) and use case (Section 5.2) requirements related to the Developer Activity Monitoring features of the CROSSMINER Eclipse IDE Plug-in and defined in the Project Requirements document (deliverable D1.1) are covered by the interim version of the plug-in that uses the final Developer Activity Monitoring features. In the last column of the tables an empty circle (○) denotes that the requirement is minimally (or not) covered, a half-filled circle (◐) denotes that it is only partially covered, and a filled circle (●) denotes that it is mostly (or fully) covered from this deliverable's point of view.

5.1 Technical requirements

D74	The IDE shall provide a settings interface to the user, where the different properties of the CROSSMINER IDE plugin (like server address and port, global settings for recommendation queries, etc.) can be checked and changed. So the user can configure the plugin.	SHALL	●
D95	The IDE shall provide full control over the collected and transferred user activity monitoring data. So the user can allow or deny the collection and/or anonymised transfer of the activity data collected from their session.	SHALL	●
D96	The IDE shall recognize, compute, and extract the following user activities, metrics, or information: frequent search expression.	SHALL	●
D97	The IDE should recognize, compute, and extract the following user activities, metrics, or information: project or file open, manipulation, close, program execution, test execution, user search patterns, working time on a file.	SHOULD	●
D98	The IDE shall be able to send developer activity data (as controlled by the user settings) to the CROSSMINER server.	SHALL	●

5.2 Use case requirements

U176	There is a strict and public strategy regarding privacy and data	SHALL	●
U177	Users cannot be identified from monitoring data	SHALL	●
U178	Monitoring is able to identify testing activities	SHOULD	●
U179	Monitoring is able to identify development activities	SHOULD	●
U180	Monitoring is able to identify errors in IDE	SHOULD	○

U181	Monitoring records the time the developer works on a given file/code element/line	SHOULD	●
U182	Monitoring of developer activity can be disabled by the developer	SHALL	●
U183	Types of data collected from monitoring are transparent to the developer	SHALL	●
U220	User and admin documentation is embedded into the UI	SHALL	●
U225	Plugin supports the latest supported release of Eclipse	SHALL	●