# CodeMetropolis: Eclipse over the City of Source Code

Gergő Balogh, Attila Szabolics, and Árpád Beszédes
Department of Software Engineering
University of Szeged, Hungary
geryxyz@inf.u-szeged.hu, szabolics@inf.u-szeged.hu, beszedes@inf.u-szeged.hu

***Abstract*—The graphical representations of software (code visualization in particular) may provide both professional programmers and students learning only the basics with support in program comprehension. Among the numerous proposed approaches, our research applies the *city metaphor* for the visualisation of such code elements as classes, functions, or attributes by the tool CodeMetropolis. It uses the game engine of Minecraft for the graphics, and is able to visualize various properties of the code based on structural metrics. In this work, we present our approach to integrate our visualization tool into the Eclipse IDE environment. Previously, only standalone usage was possible, but with this new version the users can invoke the visualization directly from the IDE, and all the analysis is performed in the background. The new version of the tool now includes an Eclipse plug-in and a Minecraft modification in addition to the analysis and visualization modules which have also been extended with some new features. Possible use cases and a detailed scenario are presented.**

***Index Terms*—Integrated Development Environment, software visualization, city-metaphor, integration.**

## I. Introduction and Motivation

The evolution of tools and methods has always been parallel with the history of professions. Software development is not an exception either. All participants have a tool set which follows their daily workflow. For developers, this tool set is the so called Integrated Development Environment, or IDE for short. Its predecessor was a simple text editor equipped with basic features, like syntax highlighting and smart text completion. As programming languages evolved and the schedule of the development process became more stressed, these tools were equipped with more features to support new ways of integration with various components and external programs. Developers are used to these convenient environments. They rarely close these programs, so probably any program has to offer some degree of integration with IDEs to fit into the daily routine of developers.

The graphical representation of the source code could provide new viewpoints which are crucial for creative work and problem solving, but the world of source code is still highly dominated by textual representation. Software visualization techniques range from simple diagrams and charts to detailed metaphors [1]. The integration of these techniques into IDEs has barely begun.

Our goal was to build a bridge between coding and visualization. We chose Eclipse among the IDEs because it was a common tool for Java developers. Software visualization is embodied by CodeMetropolis which utilizes the well known city metaphor where source code components are represented as part of a generated city. We implemented a set of plug-ins which was able to connect these two softwares, hence it became capable of integrating an elaborated visualization technique without disturbing the daily routine of developers.

These tools enable developers to launch visualization and initialize the buildings of the virtual city. To help to find the most relevant parts of the visualization, a manual and an automatic navigation were included.

The main contribution of the paper is the introduction of the integration of CodeMetropolis to the Eclipse IDE, and possible use cases. The paper is organized as follows. Sections II introduces the necessary context: related work, background concepts, and technologies. Section III is the core of our contribution, it presents our new tools which integrate CodeMetropolis into Eclipse. In Section IV and V we provide use cases and demo scenarios, respectively. Section VI is about our future plans. We refer the reader to the supplement materials in Section VII.

## II. Background

### A. Visualizing Software as Virtual City

The numerical properties of source code – called metrics – are widely used to measure software code quality and detect problems early in the development phase [2]. Several solutions exist to analyze the source code and measure various metrics on it (e. g. [3]), but finally this data has to be presented to people. One way to achieve this is to generate images or videos which will display the gathered information.

Data visualization has four phases: filtering, mapping, rendering, and displaying [4]. These are illustrated on Figure 1. In the filtering phase, raw data is processed and the data of interest is prepared. This step determines *what* we want to visualize: different entities and their properties are selected based on the goal of visualization.

In the mapping phase, the entities of the measurements and their properties are assigned to the objects of the visualization space and to their attributes. This phase decides *how* the entities and properties will be presented. Mapping is usually based on some metaphors that determine the domain and the set of objects that can be used in the visualization space. For example, the forest metaphor allows trees and tree

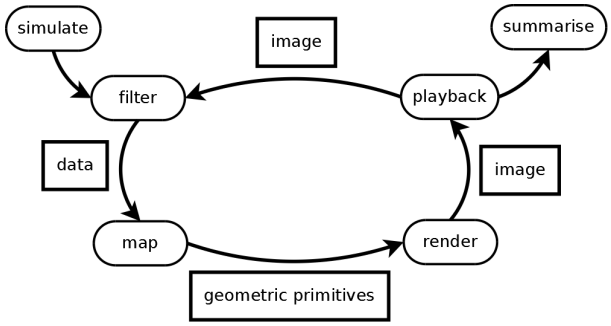


Fig. 1. Phases of data visualization

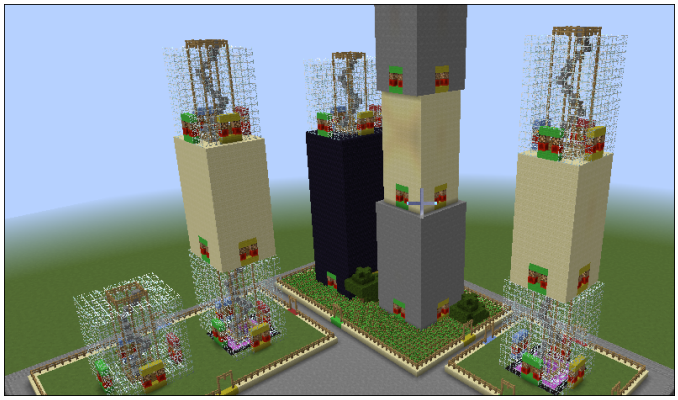

Fig. 2. Visualization of a sample project.

groups (forests) as objects, or the architectural metaphors use buildings and other architectural objects.

Rendering determines the shapes and other visual attributes of the objects and places them in the visualization space. According to the mapping, the actual attributes of the objects are determined based on the property values of the entities.

Finally, the displaying phase is responsible for presenting the filled visualization space, i. e. creating an image (or a series of images). It can be static, displaying a single image, but it is usually dynamic in the sense that the user can interactively change the view parameters by zooming, turning around, or moving the center of view.

In software visualization there are mappings between the code elements and the visualization space objects. As mentioned above, these mappings are usually based on metaphors, e. g. a forest or landscape metaphor. The metaphor determines which objects can be used in the visualization space and which attributes of these objects can potentially be used to capture the property values of the represented entities. The city metaphor [5] is one of the best known metaphors in software visualization. As it was described by Wettel and Lanza, the source code elements (usually classes or functions) are assigned to buildings and the whole software is presented as a city.

At a high level, the city metaphor assigns classes to buildings and attributes to the width, length, height, or color of the buildings. Sometimes additional elements like roof size, shape, or color are used to express some metrics. At lower levels, when the internal structure of the classes should also be presented, the buildings represent methods, and a class is represented as one version of realty. Buildings or realties are then grouped into districts according to their relationship in the code; districts are usually formed from namespaces or packages. The hierarchical structure of packages are usually represented by flat platforms that are elevated from their context.

CodeMetropolis [6], [7] utilizing the expressive power of today computer games is one implementation of this technique and metaphor. It generates a virtual city for Minecraft [8], which is a first person role playing game. After entering into the generated world, developers can explore the districts of namespaces, walk around the gardens of classes[1], then climb up on virtual staircases in the middle of methods to look upon the city of source code.

### *B. Writing Code with the Aid of Integrated Development Environment*

A set of tools is commonly used by developers during their daily routine. These tools include compilers, linkers, and text editors. The functions of these tools add up the features of the first *integrated development environments,* or IDE for short. Today, these features only provide the most basic core functionality of these all-in-one, customizable Swiss knifes of developers.

All IDEs provide a certain degree of extensibility. It is usually supported via a plug-in with *application programming interface,* API for short. These plug-ins can enhance existing features or implement new ones. Some of them provide further integration with other tools making the IDEs more robust. The complexity and quality of these extensions have a very wide range. There is a support for testing, for example the execution of the unit test written by using the well known JUnit framework. Developers can design the basic structure of the software using the UML modeling language, and generating the corresponding source code with plug-ins.

Our experience suggests that the integration of tools capable to display a visual representation of the code is fallen behind other supported features. Several attempts are made to integrate software visualization into the daily work of developers. Rob Lintern et al. [9] provide a new plug-in to integrate graphical representation. The city metaphor has also been used previously [10], [11].

## III. Integration of Eclipse and CodeMetropolis

### *A. Overview*

Our goal was to integrate a highly customizable and detailed visualization tool to a development environment. As a result, developers can get customized visual information about their system fast and without leaving their well-known environment.

[1]Gardens are grassy platforms surrounded with fences representing the classes. Each level of the buildings inside represents a member of the class.

Besides that, we would like to provide an easy way of navigation in the city to avoid wasting time on searching for the place representing the inspected part of the source code.

We have chosen CodeMetropolis as our visualization tool because it provides rich graphical possibilities and real time navigation in a fully interactive virtual world. The implementation has three interlinked components, shown in Figure 3. The first is *Eclipse,* the IDE itself, the second is *Minecraft,* which displays the generated city, and *SourceMeter* [12], a static code analyzer, which provides the metrics and the structures of the source code. These are connected via *CodeMetropolis* that converts the data to a visual representation using the given mapping and city metaphor. These are represented as components on Figure 3.

Since the 2013 release of CodeMetropolis toolkit, it has undergone significant changes. A new placing algorithm has been implemented to provide a more optimal city layout. A brand new build system has also been added which resulted 100 times faster block creation. The mapping format has been completely redesigned to provide a cleaner syntax and a lot more options. The background logic of the toolkit had to be changed at multiple points to fit our intentions. Most of these changes were done as part of the preparation for integration.

There are two small extensions, an *Eclipse plug-in* and a *Minecraft modification* (or mod for short). These are integrated into the Eclipse and Minecraft respectively, and provide communication with the parts of the CodeMetropolis toolchain. The developers interact directly with the game and the IDE.

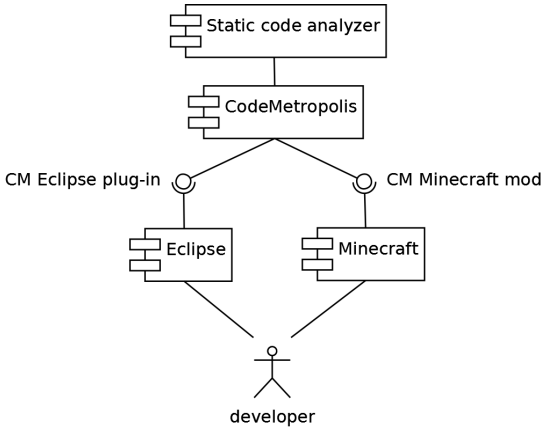


Fig. 3. Overview of integration

## B. Modification of Minecraft

Minecraft is a role playing game where the player is able to move and interact freely with the world. The world itself is built up of cubes of 1 cubic meter compared to the player itself, who is 2 meters tall. These so called blocks have various properties and features based on their material. For example, you can use *stone* blocks to build a *furnace,* then *coal* to fuel it and extract *gold ignite* from *gold ore.* There are more than 200 different types of blocks which can be placed into a realm that is bigger than the surface of the Earth.

The current version of Minecraft End User Licence Agreement [13] allows users to change the game once they have bought the license with the condition that they will not sell those changes as original features. This made possible the formation of global and local communities, whose members are continuously seeking new ways to extend the features of the game with modifications, or mods for short.

These mods can have a wide range of goals, from introducing new types of blocks or capabilities to integrating with other third party tools, like ours, the *CodeMetropolis mod.* It is a collection of recompiled Java classes which provides the following features and functions.

*1) Synchronizing:* To prevent any concurrent modification with the game, it disables the user interface while building the generated city. After the conversion the target world is reloaded. We also provide informative messages to notify the user about the state of the process. These feedbacks are shown on Figure 4.

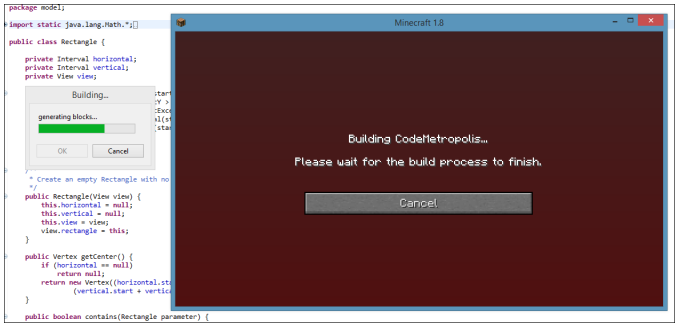


Fig. 4. Feedback during the rebuilding of the city

*2) Positioning the Player:* It allows the external processes to set the position and orientation of the player. It is used to redirect the attention of the developer to different components by automatically moving him to a new part of the city.

## C. CodeMetropolis Plug-In for Eclipse

As stated earlier, Eclipse is an Integrated Development Environment. It is one of the most commonly used tools by Java developers. Its main functions are grouped around source code editing, compiling, and running the binary code either in debug or release mode and project management. Since it is beyond the scope of this paper, we do not present an elaborated list of its features, we simply highlight the most important ones for our purposes. Starting with project management, to provide basic file, library and source code management, Eclipse utilizes the common tree view to display the structure of the program. The developer can open the file for editing by double clicking on it. Afterwards, the content of the file becomes visible in the main area. This pane supports multiple opened files by displaying them in a tab control. Functionalities are also available via toolbars and standard menubars.

All these components can be extended with third party tools called plug-ins. The plug-in infrastructure plays a key

role in Eclipse, in fact some of its basic features are also implemented as plug-ins. The API lets the external code collect information about the development process and change the layout of the graphical user interface. Our *CodeMetropolis plug-in* utilize these possibilities by detecting the name of the edited source file and adding new buttons and menu items to the GUI (Figure 5). It provides the following features, which are available via menu- and toolbar as well.

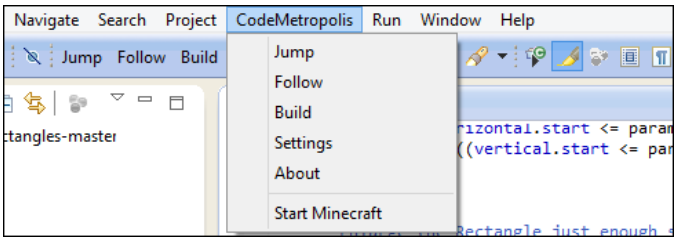


Fig. 5. Graphical User Interface of the CodeMetropolis Eclipse plug-in

*1) Building:* This functionality initiates a complete rebuild of the source code. During this process, the code is analyzed with SourceMeter, and the result is forwarded to the CodeMetropolis toolchain which generates the city and renders it with the help of Minecraft. The user is continuously noticed about the state of the conversion. In the current version, the developer has to initiate the building manually, because the time it takes highly depends on the size of the codebase.

*2) Jumping:* The size of the generated city could be too large to manually search points of interest. To overcome this, our plug-in lets the user quickly navigate to the building representing the currently open and active file by using the jump feature. With this, the developer can spend more time with the true exploration of the source code without clueless wandering.

*3) Following:* We also provide an automation over the jumping function, called following. When users turn this feature on the system will be continuously checking the open and active file, and update the position of the player accordingly. It means that the player will always be near the building representing the currently edited file.

*4) Changing the Settings:* The integrated tools required some basic configuration. These contained the location of the SourceMeter and Minecraft, and also the path to the mapping file of CodeMetropolis which specifies the meaning of the visual attributes in the city.

## IV. Use Cases

We have identified two major use cases for our tool. Exploration tasks consist of the actions that needs to be performed to comprehend source code which was written by someone else. Our assumption is that developers need to execute these tasks during their daily routine. The other potential use of the tool is in education. Visual analogies can make learning a lot easier for most of the students.

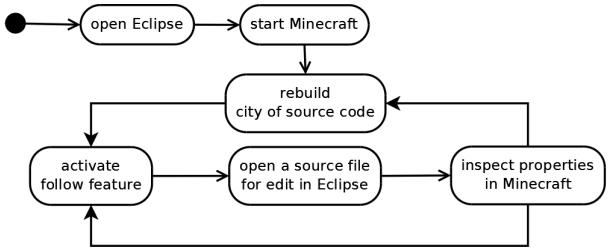


Fig. 6. Overview of exploration use case

### A. Exploration Task

Developers, either juniors or experts often have to join ongoing projects. In these, the code base already contains the implementation of some features. The size, the importance, and the quality of these show a wide range of variation. The developers have to find the location of the important parts of the code and need to gather general knowledge about the properties of various code entities, like classes and methods. In other words, navigating confidently trough the code base can speed up the implementation of further features.

The integration can help to explore the code by combining an intriguing and rich visual representation with the familiar environment of the Eclipse IDE. The use case begins with opening of Eclipse and then launching the Minecraft right from the CodeMetropolis menu. The next step is to generate the virtual city which is going to represent the source code. To do this, developers use the Build feature of the CodeMetropolis Eclipse plug-in. Afterwards, the users are able to open the generated world in Minecraft and begin the exploration task itself. This usually contains a series of repeated steps, during which various code entities are inspected. Activating the Follow feature in Eclipse ensures that the player is always in the garden which represents the actually edited class. This synchronous navigation lets the developers compare the values of source code metrics which are difficult to see from the code, but displayed as various visual properties of the buildings in Minecraft. This might be less tiring than manually comparing a bunch of raw metric values, especially in case of large systems. Usual steps of exploration are shown on Figure 6.

### B. Education

For students, it usually takes much time to fully understand the concept and advantages of object-oriented design. They need to learn a new perspective on programming tasks to be able to properly design the structure of their systems. By visualizing the structural parts of code, they can see programs in a new way. They can comprehend the structure of the source code just by walking around in a virtual metropolis. The relationship of packages, classes, methods, and attributes can easily be presented through the buildings of the city. They can understand underlying properties (metrics) and their connections. This kind of visualization is also a great way to present programming to younger children. Real life analogies can make them feel more comfortable while talking about abstract things like classes or metrics.

During a learning session, students should perform the following actions. First, they need to start both tools: the IDE and the game. After opening the selected project, they are able to build the virtual city and enter into the visualization. Then, they should investigate and understand the connections between the objects of the city and the code entities. Jump function can be very useful during this phase. Implementing new features or modifying existing ones affect the structure and quality of the code. It is recommended to rebuild the visualization after the changes so students can examine the effects of their actions. Our assumption is that by repeating these steps multiple times during the life cycle of the project they can monitor their coding quality and recognize structural weaknesses in the system.

## V. Demo Scenario

In this section, we present two demo scenarios. Both of these use the open-source project *tutorial-refactoring-rectangles* [14] which is a small Java program. It serves as a classroom exercise for students to practice various refactoring techniques. To help the reader understand the scenarios, we only specify the relevant metrics and properties, however the full mapping can be found in the supplement materials as sample mapping file. The elements of the source code are assigned to the same building in all cases, so classes are represented as gardens, their methods are displayed as the floors of buildings, and the stone plates stand for namespaces or packages. However, the properties of those which are linked to various metrics are different and will be explained later. More demo scenarios can be found in the supplement material.

### A. Inspecting Various Visual Properties of a Single Building

In the first scenario, the *logical lines of code* are mapped to the *height of the floors*, and the *number of statements* is visualized as the *material the walls are made of*. This means that if a method has more lines which are neither empty nor comment, the related floor will be higher. On the other hand, if a method contains less statements, the floor will be built from lighter materials like sandstone or glass instead of the darker ones like stone or obsidian. The minimal height of a floor is 9 blocks and the materials range from glass to obsidian.

Figure 7 shows the constructor of the Rectangle class and its visualization. The assigned elements are highlighted and connected. In this case, the code from line 41 to line 46 is represented with the floor in the middle. This is made from sandstone and it is neither extremely tall nor short. Its visual appearance suggests that it cannot contain too many logical lines of the code or statements. Furthermore, the lightness of the material and the moderate height indicate that the value of these two metrics are relatively close to each other. The implementation of the constructor contains 4 logical lines and 4 statements. It means that the average ratio is 1 statement per line.

These values of metrics can be calculated or compared manually but the time it takes depends on the size and the complexity of the code. The use of CodeMetropolis as visualization and its plug-in for Eclipse could speed up this process by providing a rich and interactive visual representation. Inspections like this are the core step when the developer needs to explore new source code.


Fig. 7. Inspection of the constructor of the Rectangle class and its visualization

### B. Compare Two Different Buildings

In this case, two buildings are compared, namely two methods of the *Rectangle* class: the *contains* and the *equals* methods. We use the same ranges for building material as in the previous case. The *number of statements* is represented with the material, but the height is assigned to the *cyclomatic complexity* used to indicate the complexity of a program. It is a quantitative measure of the number of linearly independent execution paths through the source code of a program.

Figure 8 shows the relevant parts of the source code and its graphical representation. The assigned elements are highlighted and connected. In this case, the code from line 55 to line 69 and from line 91 to line 99 are represented with the two stone floors. Both of these are made of the same material, so they contain similar amount of statements. The one above is higher, so it is more complex than the other.

To decrease the overall complexity of the class and improve readability, developers may refactor the most complex parts of the class, in this case the *contains* method. After finding the relevant method, the modification can be applied in Eclipse. In this case, the last two lines (line 67 and 68) can be extracted into two new methods, which will compare the vertical and the horizontal size of the rectangles. Then, the developers rebuild the virtual city to see the affect of their changes. These steps can be repeated to accomplish the required refactoring tasks.

## VI. Future Work

We plan to add new features to make the use of the tool more easier. One of these features is to make navigation two-

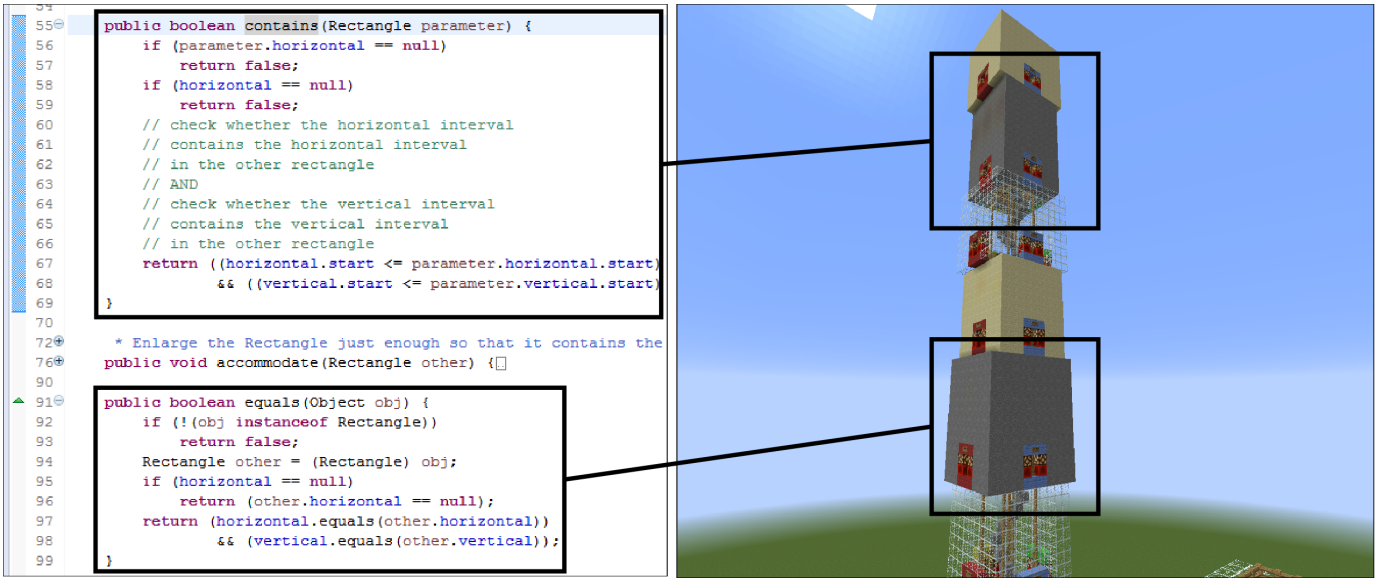


Fig. 8. Comparison of two methods and their corresponding floors in the virtual city

way, so the focused element in Eclipse would change whenever the player enters a new area in the city. Furthermore, we want to add new analogies to make a clean representation of the connections between the classes. A mapping file editor is also on the way to make it possible to create mappings without manually changing configuration files. We also plan to make further steps to improve the building speed and add an auto-build function that detects the changes an rebuilds the metropolis if needed.

With these improvements we think CodeMetropolis might be useful outside the classrooms. For example, it could help developers during refactoring sessions when neither new features are implemented nor bugs are fixed, only the underlying structure of the code is changed to improve the quality of the software. With the proper mapping of metrics it might help to detect the possible weak spots of the system.

These features will be evaluated during an elaborated user study.

## VII. Supplement Materials

Both the CodeMetropolis Eclipse plug-in and the CodeMetropolis Minecraft mod are freely available at our GitHub site:

`https://github.com/sed-szeged/codemetropolis/releases/tag/CMEclipse_1.0.1_beta`

This package contains the binary distribution (compiled for Windows), a detailed documentation, and a sample mapping file for our tools. The guide to the setup and installation is in `docs\install.html` and `docs\eclipse.html` files.

For further demo scenarios and tools in action we refer the reader to our YouTube channel:

`https://youtu.be/A0nZKu9Zsjg`

## References


[1] S. Diehl, *Software visualization: visualizing the structure, behaviour, and evolution of software*. Springer Science & Business Media, 2007.

[2] T. Gyimóthy, R. Ferenc, and I. Siket, "Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction," in *IEEE Transactions on Software Engineering*, vol. 31, no. 10. IEEE Computer Society, Oct. 2005, pp. 897–910.

[3] R. Ferenc, Á. Beszédes, M. Tarkiainen, and T. Gyimóthy, "Columbus – reverse engineering tool and schema for C++," in *Proceedings of the IEEE International Conference on Software Maintenance (ICSM 2002)*. IEEE Computer Society, Oct. 2002, pp. 172–181.

[4] C. Upson, T. A. Faulhaber Jr, D. Kamins, D. Laidlaw, D. Schlegel, J. Vroom, R. Gurwitz, and A. Van Dam, "The application visualization system: A computational environment for scientific visualization," *Computer Graphics and Applications, IEEE*, vol. 9, no. 4, pp. 30–42, 1989.

[5] R. Wettel and M. Lanza, "Visualizing software systems as cities," in *Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007. 4th IEEE International Workshop on*. IEEE, 2007, pp. 92–99.

[6] G. Balogh and Á. Beszédes, "CodeMetropolis - code visualisation in Minecraft," in *Proceedings of the 13th IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM'13), Tool Track*, Sep. 2013, pp. 127–132.

[7] ——, "CodeMetropolis - a Minecraft based collaboration tool for developers," in *Proceedings of the 1st IEEE Working Conference on Software Visualization (VISSOFT'13), New Ideas or Emerging Results track*, Sep. 2013, pp. 1–4.

[8] "Minecraft Official Website." [Online]. Available: http://minecraft.net/

[9] R. Lintern, J. Michaud, M.-A. Storey, and X. Wu, "Plugging-in visualization: Experiences integrating a visualization tool with eclipse," in *Proceedings of the 2003 ACM Symposium on Software Visualization*, ser. SoftVis '03. New York, NY, USA: ACM, 2003, pp. 47–ff. [Online]. Available: http://doi.acm.org/10.1145/774833.774840

[10] A. Bacchelli, F. Rigotti, L. Hattori, and M. Lanza, "Manhattan-3d city visualizations in eclipse," *ECLIPSE IT*, vol. 2011, p. 307, 2011.

[11] A. Biaggi, "Citylyzer-a 3d visualization plug-in for eclipse," Ph.D. dissertation, Bachelor's thesis, University of Lugano, 2008.

[12] "SourceMeter Official Website." [Online]. Available: https://www.sourcemeter.com/

[13] "Minecraft End User Licence Agreement." [Online]. Available: https://account.mojang.com/documents/minecraft_eula

[14] "Small Refactoring Classroom Exercise Website." [Online]. Available: https://sewiki.iai.uni-bonn.de/private/daniel/public/tutorials/small_refactoring