CodeMetropolis – code visualisation in MineCraft

Gergő Balogh and Árpád Beszédes

Department of Software Engineering
University of Szeged
Hungary
{geryxyz, beszedes}@inf.u-szeged.hu

***Abstract*—Data visualisation with high expressive power plays an important role in code comprehension. Recent visualization tools try to fulfil the expectations of the users and use various analogies. For example, in an architectural metaphor, each class is represented by a building. Buildings are grouped into districts according to the structure of the namespaces. We think that these unique ways of code representation have great potential, but in our opinion they use very simple graphical techniques (shapes, figures, low resolution) to visualize the structure of the source code.**

On the other hand, computer games use high quality graphic and good expressive power. A good example is Minecraft, a popular role playing game with great extensibility and interactivity from another (third party) software. It supports both high definition, photo-realistic textures and long range 3D scene displaying.

Our main contribution is to connect data visualisation with high end-user graphics capabilities. To achieve this, a conversion tool was implemented. It processes the basic source code metrics as input and generates a Minecraft world with buildings, districts, and gardens. The tool is in the prototype state, but it can be used to investigate the possibilities of this kind of data visualisation.

I. Introduction

Software systems could reach virtually infinite complexity by their nature. In theory, there is no limit of control flow embedding, or the number of methods, attributes, and other source code elements. In practice, these are bound to the computational power, time and storage capacities. To comprehend these systems, developers have to construct a detailed mental image. These images are gradually built during the implementation of the system.

Often, these mental images are realised as physical graphics with the aid of data visualisation software. For example, different kinds of charts are used that emphasize the difference among various measurable quantities of the source code, or UML diagrams which are able to visualise complex relations and connections among various entities in the system.

A. Classical visualisation

People are different, each of them having their own point of views. They use various "tools" to comprehend the world. Some of them need numbers, others use abstract formulas, but most of them need to see to visualize the information as colours, shapes, and figures. To fulfil the expectations of people, a lot of data visualization techniques and tools were designed and implemented. It exceeds the purpose of this article to exhaustedly evaluate these techniques and tools, but in our opinion traditional visualisation tools like Rigi [11], sv3D [5] and SHriMP Views [8] are built on innovative ideas but often it is difficult to interact with them, and they usually fall behind in terms of graphics from today's computer games, for instance.

978-1-4673-5739-5/13/$31.00

B. About CodeCity and EvoSpace

The most closely related approaches to our tool are CodeCity [10] and EvoSpace [4] which use the analogy of skyscrapers in a city. CodeCity simplifies the design of the buildings to a box with height, width, and colour. The quantitative properties of the source code – called metrics – are represented with these attributes. In particular, each building represents a class where height shows the number of methods, width shows the number of attributes, and color shows the type of the class. The buildings are grouped into districts as classes are tied together into namespaces. The diagram itself resembles to a 3D barchart with grouping. EvoSpace uses this analogy in a more sophisticated way. The buildings have two states: closed – when the user can see the large scale properties like width and height, and open – when we are able to examine the low, small scale structure of the classes, see the developers and their connections. It also provides visual entity tagging and quick navigation via the connections and on a small overview map.

Despite their appealing appearance and great potential in general, these tools still use relatively low fidelity graphics compared to today's most advanced computer games. In this paper we introduce our approach for visualising source code using the same metaphor but employing a sophisticated game engine called Minecraft.

C. About Minecraft

Minecraft [6] is a popular role-playing game. It is written in Java language and uses OpenGL graphical engine to display the scenes. Both of these technologies are widely supported on major platforms. It is distributed both as free and as commercial software with support. There are several binary formats used to describe the game scene – called world – which are either open standards or free formats.

The game itself does not have a strict game-flow. Its main focus is creativity and the joy of creation. Only the available computation power and the storage capacity can limit the fantasy of the player.

Figure 1. JUnit project visualized by CodeMetropolis

The main concept in the game is the block. It is a box with about one meter long sides, compared to the player. Almost everything is built up of it, so the whole World is a 3D matrix filled with blocks of various types. The player can collect the blocks, create (craft) new ones and interact with them. The game is similar to a virtual Lego with infinite playground and an infinite number of building blocks.

The player has its own backpack which can be filled with blocks and tools. The tools are used to accomplish or speed up various tasks like mining. Every tool has its own properties. There are some common materials like dirt, stone and sand, and some special blocks have more sophisticated purposes, for example cheast, which can store items, and various electronic (red wire) devices like pressure sensitive plates, buttons, and switches.

Due to its extensibility, its simple yet sophisticated functions, and its rich palette of possibilities Minecraft can display complex structures with a low overhead.

II. Data visualisation in CodeMetropolis

CodeMetropolis is a command line tool written in C#. It takes the output graph of Columbus Tool [1] and creates a Minecraft world from it. The output is given with a unique binary format, but the related tools and the format itself are under development and not yet published. For these reasons we could not give a detailed specification of the output format. The world uses the metropolis metaphor, which means that the source code metrics are represented with the various properties of the different kinds of buildings. Figure 1 shows an example world.

The representation has two main levels. On the data level, each entity has its own property set – for example metrics. In the current version, these are loaded from the previously mentioned graph, but we plan to support other data sources, for example XML files. These data are displayed on the metaphor level. All buildings in the metropolis belong to this. The buildings and the world itself has a couple of attributes which control visual appearance. The properties are mapped to the attributes in order to visualise the data. However, in the current version this mapping is hardcoded, the further versions will support customisation with a sophisticated mapping language.

A. Considered metrics and properties

As mentioned before, source code metrics were used to express the various properties of the system. Our source code analysis toolset, Columbus, produces a number of different source code metrics for various languages. Some of the most commonly used metrics are the following:

NOI Number of outgoing invocations, **NII** Number of incoming invocations, **LOC** Lines of code, **TLLOC** Total logical lines of code, **LLOC** Logical lines of code, **NUMPAR** Number of parameters, **NL** Nesting level, **McCC** McCabe's cyclomatic complexity, **NOS** Number of statements.

These common metrics were considered during the experiments, however the current version of CodeMetropolis does not use all of them (we detail the used ones below).

To create a visualization with sufficient expressive power, the structure of the system has to be displayed as well. The graph input contains this information, expressed with the edges of the graph. From several types of edges only the containment relation was used.

Figure 2 shows a simple input graph. It represents a small Java program which only consists of a couple of source files. The source code and the graph of this example were included in the sample inputs.

B. Data mapping

The current version of the converter uses the following entities and attributes to visualise the source code. These items

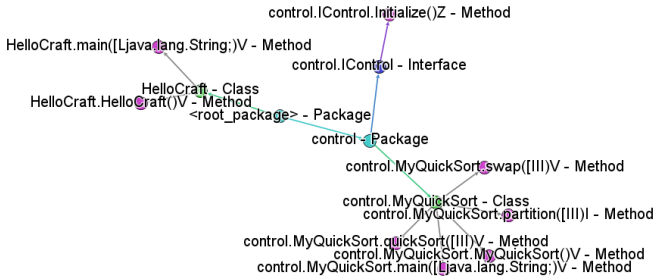

Figure 2. Simple graph input

are highlighted on Figure 3.

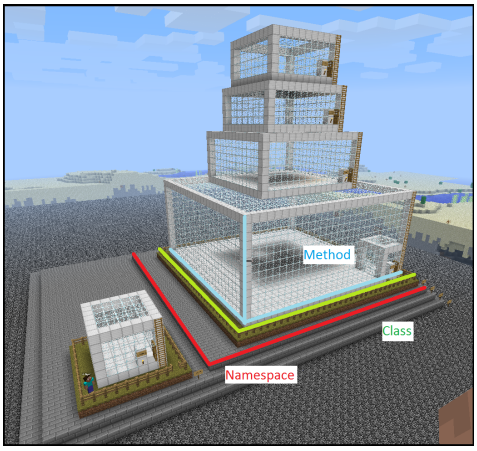

Figure 3. Items of the metaphor level

- A Plate is able to group various type of entities including other plates. It is used to display the namespace hierarchy like a tree-map. In the generated world, it is displayed as a solid rectangle of stone blocks. Its width and length were adjusted automatically to fit to its contents. Attributes:
  - Name
- Districts are similar to Plates. It is used to represent individual classes. It is displayed as a plate of grass blocks surrounded with fences. Attributes:
  - Name
- A Building is another compound entity. It consists only of floors which are placed on the top of each other ordered according to their width and length. The converter uses this entity to group floors, however it has no meaning on the data level. Attributes:
  - Name
- Floor (Figure 4) is a simple box of glass blocks with iron lattice and bottom. It represents a single method. Its width and length are mapped to McCC and its height represents the size of the method in the logical lines of the code. Attributes:
  - Name
  - Width
  - Length
  - Height

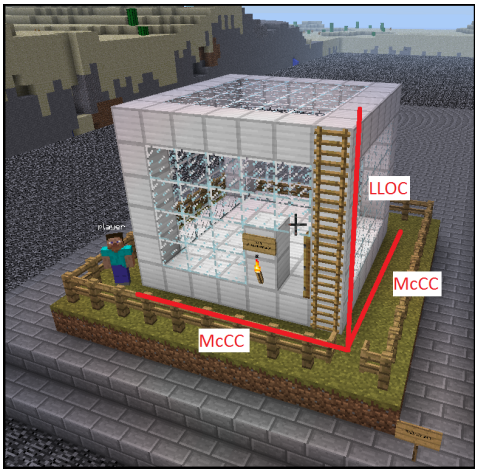

Figure 4. Attributes of a floor

All of these entities have a minimal size, to let the player walk in and are labelled with post or wall signs showing the name of the represented source code item (Figure 5). Because these sign could be seen only from short distance, we plan to provide some location information via a mini-map of the city.

Figure 5. In-game labels

Figure 6 shows a concrete class from JUnit represented with a single building on a plate. The values of the mapped metrics are shown as well, which are normalized. The values are scaled between a minimum (4 blocks) and a maximum (25 blocks) values. The source code of this class can be seen in Listing 1.

```
public class InvokeMethod extends Statement {
    private final FrameworkMethod fTestMethod;
    private Object fTarget;

    public InvokeMethod(FrameworkMethod testMethod,
        Object target) {
        fTestMethod = testMethod;
        fTarget = target;
    }

    @Override
    public void evaluate() throws Throwable {
        fTestMethod.invokeExplosively(fTarget);
    }
}
```

Listing 1. Source of `InvokeMethod` class

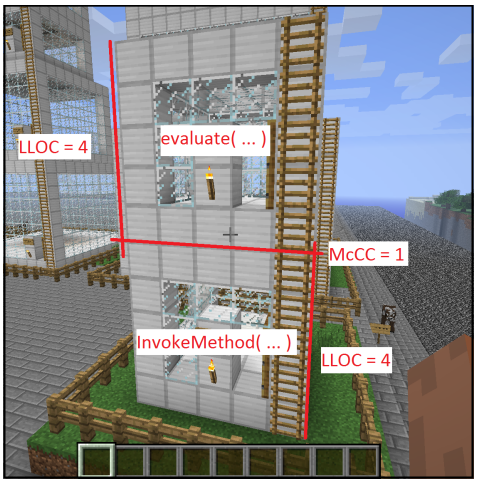

Figure 6. Example class visualization

C. Multilevel data visualisation

One of the problems of understanding complex systems is the flow of information. In this case, the users have too much low level information and they are unable to filter out the irrelevant data. To overcome this problem, we introduced multilevel data visualisation. With this technique, the users have the opportunity to decide how much details they would like to see.

In Minecraft, this is simply done by looking at the entities from different distances. Since Minecraft allows one to fly in a so called creative mode, the user can inspect the metropolis from the bird's eye view thus seeing the large scale size of the classes and the namespaces. Then, one can walk on the streets and compare the size and the complexity of the methods. It is even possible to go inside the methods and – in the future version – explore their inner structure represented with furniture.

III. Benefits and Use cases

The usefulness of a tool like CodeMetropolis depends on various factors including the experience and personal values of the users. People who naturally use similar metaphors to understand the world this could be a straightforward way of visualisation, while for many others the approach would be merely an interesting but generally an experimental idea.

With these considerations kept in mind, a couple of use cases will be explained. There are two main applications of the data visualisation tools – and CodeMetropolis. It can be used on real life projects, or in classrooms, to help students understand complex relations and concepts. In the use cases, we were focusing on the first one, but the corresponding use cases of the second one can be easily found.

A. Bad smells detection

Bad smells are the parts of the code which will probably cause an error or are difficult to maintain. [2] Developers usually identify bad smells by reading the source code.

With CodeMetropolis, developers are actually able to see some of the smells. For example, if the logical lines of the methods are mapped to the height of the floors – as in the current version – a giant class can be easily found by searching the tallest building of the metropolis. Then, by walking or flying around this building the count and size of the methods could be examined. If there are only a couple of them, these are large methods that need to be split up. On the other hand, when there are a lot of small methods, the class might be a so called god or brain class which needs to move its functionalities into another class, maybe a new one.

B. Inspect the structure of code

When new developers are assigned to an ongoing project they have to understand the large scale structure of the system. They usually browse via many directories and source files, or in better cases they look at the different diagrams and documentations of the system. But any case simply browsing through a myriad of code and documents is tedious and the developers' motivation rapidly decreases. Furthermore, documentations is often outdated or incomplete.

The worlds of CodeMetropolis are generated from the source code, so they always reflect the current state of the system. Developers can fly over the metropolis and see the clusters of the classes and the namespaces.

C. Annotating entities

When developers are inspecting the source code, they often leave comments to mark its parts.

The future version of our conversion tool will support code annotation. When developers put a wall or post a sing on some entities (floors, buildings), the text on it will be inserted into the source code as a comment. Furthermore, in the multi-player mode developers can see and interact with each other, so some parts of a code review meeting can be held in the Minecraft world.

D. Present code history

There are several open-source Minecraft servers which support extendibility. We plan to use these servers to visualize the code history gathered from the version controlling system.

With the use of historical data, representations can be generated for each revision, then several computer controlled players will be used to literally build these states of the metropolis. Each player represents a developer and will build the parts of the buildings that they coded. For example, if a developer inserts a new method into a class, the player will go to the corresponding building and build a new floor representing the new method. Project managers and other developers can join the server and see the evolution of the system.

E. Identify untested code

Test coverage data will be integrated into a future version of the converter. For example, torches and glowstones can be used to illuminate the building representing the tested parts of the code. The height of the fences around the classes can represent the number of test cases associated with the given class. The colour of these fences can be used to indicate the result of the test case (pass, fail).

F. Understanding inter-metrical relations

Since source code can be complex having many different properties and attributes and relations to other entities, classical visualisation techniques like graphs and charts cannot represent so much information at once. However, to understand the relation and connection among the various metrics, the global context has to be analysed.

To address this problem, CodeMetropolis will use various sophisticated metaphors. For example, a floor represents a method. Its width and length are mapped to its complexity and its height indicates its size. Furthermore, the number of windows and doors visualise the count of its parameters. There are torches on the wall if the method is tested and the fences around it indicate the number of passed test cases. Even if the developers do not know the formal definition of these metrics, they are able to see the consequences of their actions while writing the code. Sooner or later, they will assign informal meanings to the different kinds of graphical elements in the metropolis and they will perceive the represented source code as a whole.

IV. Current state

CodeMetropolis is in a prototype state, and we have a lot of plans for various additional functionalities to be implemented. However, with this paper we intend to publish current results and further ideas as a proof of the implementability of the concept.

This version was written in C# using the .Net framework. It is a command line tool which takes the previously mentioned graph as input and creates a Minecraft world from it. There are a couple of open-source API-s for every major language which support editing or creations over these worlds. Our tool uses the Substrate [9] library for .NET Framework.

Currently, only the previously mentioned complexity and size metrics are visualised with the entities listed in Section II-B and only the namespaces, classes and methods are represented. However, even with this limited toolset we were able to visualise complex, real life systems and gain useful insights into the systems. As an example, Figure 1 shows the view of the JUnit [3] metropolis.

A. User feedback

> "It makes software metrics such fun that you want to do it." *an user*

We performed a number of interviews among our colleagues that included university lecturers, students and developers working on industrial code, altogether six users was asked. We wanted to gain early feedback on the converter and the visualisation technique in general. In this section, both the negative and the positive impressions about CodeMetropolis will be summarised.

The negative opinions are grouped into several topics. The first of these concerns is the problem of great distances. The metropolis of a large or medium scale project can be huge and the players need a lot of time to navigate in it. We plan to solve this problem by implementing a quick map and a navigation system. These will be included in the multi-player server enabling the players to see their location and quickly teleport to other places.

Other opinions were concerned about the learning curves, either about Minecraft or about the visualisation. In our opinion, Minecraft has very simple control logic and in-game physics, no cryptic keyboard short-cuts, or complex machinery. It can be learned quickly and easily while playing or with the use of sophisticated online resources [7] covering every detail. Minecraft supports two special items among others: cheats, which can store other items and book them, and quill, which allows the players to take notes and write books in the game. CodeMetropolis will also support these to create in-game explanation notes for the items of the metaphor level.

The last problem was the lack of simultaneous data visualisation. The users could identify only three attributes: width, length, and height. This limited set is not enough to visualise the complex items of the data level, but as explained above we will be able to extend the tool with additional properties easily.

The positive impressions are also grouped into categories. The first of these is about the expressive power provided by the metaphor and the in-game logic of Minecraft. The second one is that the "work while you play" approach can maintain or even increase the motivation of the users, especially students. Finally, the users mentioned one of our further plans, the round-trip source code management. For example, if the players destroy a floor in the metropolis, the corresponding method will be removed from the source code.

V. Future works

In this section, we will enumerate some of our further plans. Since almost all of these were explained in depth in the previous sections, here only a short list is given.

Extending the palette of the entities and attributes The future version of our converter will use an extended palette of the blocks supported in Minecraft. For example, flowers to decorate beautiful code and zombies (hostile creatures) to indicate bad practices.
Navigation support We plan to implement a mini-map and a teleportation system. The related classes will be connected with railways allowing the users to navigate and see the connections.
Round-trip source code management The changes between the source code and the metropolis will be propagated to each other.
In-game explanations Post and wall sings and books will be used to explain the meaning of the various attributes and to show the source code of the corresponding element.
Visualize source code history The functionality of open-source multi-player severs will be extended to visualize source code history. For example, computer controlled players (npc-s or bots) will build the metropolis as the developers commit their changes into the version control system.

VI. Conclusion

In some cases, developers need to step away from the source code and inspect the system from a different perspective. We believe that CodeMetropolis will be able to

maintain motivation without sacrificing productivity thanks to its intuitive and, for many people, already known graphical surface. The provided metropolis metaphor has enough expressive power to represent the complex items of the source code. Combined with high quality graphical techniques provided by today's computer games, it is able to offer a rich graphical interface, an easy to learn controlling, and a rich user experience. It is probably easier to fit in classrooms than in a commercial project. However, we will continue its development to integrate the functionalities which are useful for developers, for students, and for teachers.

Appendix A
Demonstration

The demonstration will be carried out live, which means that a small example project written in Java will be visualised from the source code to the metropolis in Minecraft. We will go through the following steps:

1) The problem will be presented in a couple of slides to warm up the audience.
2) A sample world will be generated by Minecraft to illustrate the possibilities of the game.
3) A small sample code will be introduced.
4) The example project will be converted to a Minecraft world.
5) Every attribute mentioned in this paper will be explained on the metropolis generated in the previous step.
6) The authors and the audience will explore together the metropolis of JUnit.

Appendix B
Executables and samples

The current version of CodeMetropolis can be downloaded from the following url: http://www.inf.u-szeged.hu/~geryxyz/code-metropolis.html. The published package contains the executables and two sample projects: JUnit and HelloCraft, both of them were mentioned in this paper, together with sample inputs and outputs.

The tool is distributed in two ways: in portable binaries and in a setup package. It requires the 4.5 version of the .NET framework. After installing or copying the executables into the desired location, the conversion can be started with the `CodeMetropolis.exe <input-file>.graph` command from the command-line prompt. It will prompt to press the Enter-key when the conversation is finished. The results will be produced in the CodeWorld directory under the current directory. It will contain the generated Minecraft world which needs to be copied to `<user-home>\AppData\Roaming\.minecraft\saves` under Microsoft Windows 7. Then the user will be able to open it with Minecraft as a usual world. The tool was tested with 1.5.2 version of Minecraft.

References

[1] Rudolf Ferenc et al. "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.

[2] Martin Fowler et al. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, 1999, p. 464. ISBN: 0201485672.

[3] *JUnit Official Website*. URL: http://junit.org/.

[4] D Lalanne and J Kohlas. *Human Machine Interaction: Research Results of the MMI Program*. 2009.

[5] A. Marcus, D. Comorski, and A. Sergeyev. "Supporting the evolution of a software visualization tool through usability studies". In: *Proceedings of the 13th International Workshop on Program Comprehension* (May 2005), pp. 307–316. DOI: 10.1109/WPC.2005.34.

[6] *Minecraft Official Website*. URL: http://minecraft.net/.

[7] *Minecraft Wiki - The ultimate resource for all things Minecraft*. URL: http://www.minecraftwiki.net/wiki/Minecraft_Wiki.

[8] Margaret-anne Storey, Casey Best, and Jeff Michaud. "SHriMP views: an interactive environment for information visualization and navigation". In: *CHI '02 extended abstracts on Human factors in computing systems - CHI '02* (Apr. 2002), p. 520. DOI: 10.1145/506443.506459.

[9] *substrate-minecraft A .NET SDK for editing Minecraft worlds*. URL: https://code.google.com/p/substrate-minecraft/.

[10] Richard Wettel and Michele Lanza. "CodeCity". In: *Companion of the 13th international conference on Software engineering - ICSE Companion '08*. New York, New York, USA: ACM Press, May 2008, p. 921. ISBN: 9781605580791. DOI: 10.1145/1370175.1370188.

[11] Kenny Wong. "Rigi user's manual". In: *Department of Computer Science, University of Victoria* (1998).