CodeMetropolis – a Minecraft based collaboration tool for developers

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 fulfill 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 have high expressive power. A good example is Minecraft, a popular role playing game that supports both high definition, photo-realistic textures and long range 3D scene displaying. Additionally, it provides great extensibility and interactivity for third party software. In this paper, we introduce our mission to create a virtual world of source code in which developers and other stakeholders could explore and evaluate their project collaboratively in a virtual Minecraft world. Code properties are represented by graphical primitives offered by the game engine, and various interactivity features are planned. Besides challenges of the implementation there are some fundamental research issues considering the selection of a set of visual element and mapping to source code properties. These elements have to be compatible not only with the visualisation and with the data model but also with the thinking of developers.**

***Index Terms*—Source code visualization, game engine, Minecraft, source code metrics.**

I. Introduction

The importance of visualisation techniques is undeniable in all field of science, and also in various software engineering activities including comprehension and collaborative exploration. Diagrams, charts and other graphical elements are often used to present quantitative and qualitative properties and their relations. These tools use simple and abstract graphical primitives which could not be found in real world like straight lines, points and circles. They are able to express some attributes of the software successfully but are less useful to present more complex, many dimensional contexts. There are several visualisation tools which use real life metaphors like skyscrapers to address this problem. On the other hand, computer games use high quality graphic and good expressive power. A good example is Minecraft, a popular role playing game that supports both high definition, photo-realistic textures and long range 3D scene displaying. Additionally, it provides great extensibility and interactivity for third party software.

Another motivating fact is that software developers use various collaborative tools and techniques from e-mails trough version controlling to code review tools, and most of these use the previously mentioned simpler visualisation techniques. These tools offer important but complex functionalities so they often require continuous concentration from the users. To improve the work flow, further collaboration tools are often needed that provide rich user experience and hence increase user productivity.

Our long term goal with the presented research is to join the techniques of data visualisation [**Maletic2002**], graphical game engines and collaborative tools. Our tool CodeMetropolis is a Minecraft based visualisation tool. Figure 1 shows an example of our current visualisation possibilities in it. Our tools use a popular game engine Minecraft to visualize source code, and provide additional functionality using which users will be able to collaborate with each other via a multi-player game sever.

To our knowledge, game engines have not yet been used for the purposes of software data visualisation and user collaboration.

II. Related works

People are different and use different mental processes to comprehend the world. Some of them need numbers, others use abstract formulas, but most of us like to see the information visualised as colours, shapes, and figures. A lot of data visualization techniques and tools were designed and implemented in software engineering research and practice. It exceeds the purpose of this article to exhaustedly evaluate these techniques and tools, but in our opinion traditional visualisation tools like Rigi [**Wong1998**], sv3D [**Marcus2005**] and SHriMP Views [**Storey2002**] 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.

The recent growth of web based applications and the popularity of mobile devices made it possible for the collaborative tools to reach the general public. Besides general applications like Google Drive and Facebook there are several services with

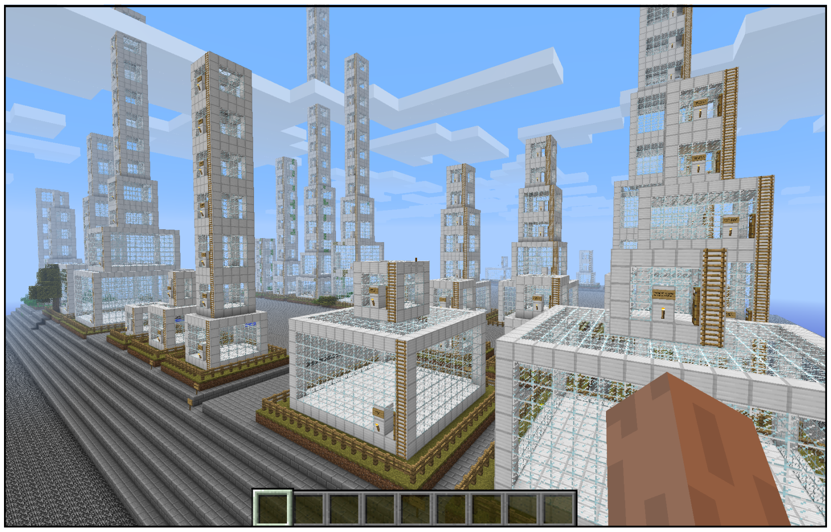

Figure 1. JUnit project visualized by CodeMetropolis

more specific purposes to aid software engineering processes like SourceForge and Github. There already exist a number of sophisticated software tools that are able to visualize the huge amount of data collected by these tools, for instance, Gource [**gource-webpage**], Logstalgia [**logstalgia**] and Star-Gate [**Ma2008**]. However, most of these tools use abstract shapes and simple graphical primitives like charts and vertex graphs.

The most closely related approaches to our tool are CodeCity [**Wettel2008a**] and EvoSpace [**Lalanne2009**] 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 colour 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. Furthermore, little collaborative features are provided that enable a simultaneous work flow of different project members. In this paper we introduce our approach for visualising source code using the city metaphor as well, but employing a sophisticated game engine and advanced collaborative features.

A. *About Minecraft*

Minecraft [**minecraft-website**] is a popular role-playing game. 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.

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.

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

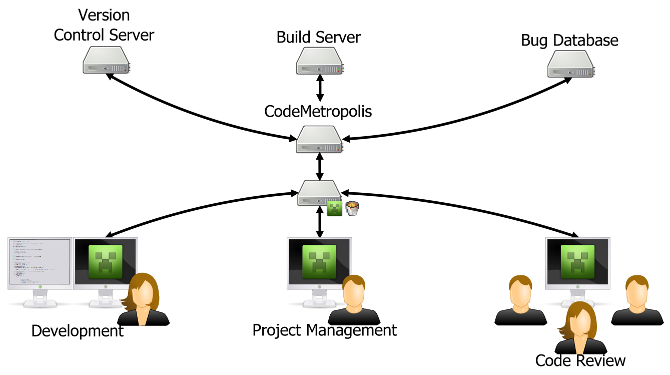

Figure 2. Scenarios and data flow

III. Visualization and collaboration in virtual world

As mentioned, motivation of project team members plays a key role when data visualisation and collaborative tools and techniques are to be used. A possible way to increase motivation is by providing rich user experience of high quality graphics provided by game engines like Minecraft. Many games support a multiplayer mode, which is in essence a collaboration platform between different users. These facts and the highly advanced extendibility of Minecraft made it possible to use it as a collaboration and visualisation tool for software developers.

Our main concept and some planned usage scenarios are shown in Figure 2. Data sources are listed on the upper part of the figure, while users and scenarios are located on the bottom part of it. They are connected via our central converter and management tool *CodeMetropolis* and backed up by a multiplayer sever of Minecraft. Users are able to interact with various parts of the world and with each other as well, and these actions can be propagated to the original data sources like version control servers and bug databases. In this manner the game could be used to inspect source code attributes during development, follow the evolution of the system by mangers or find and mark bugs during code reviews, for instance. Some of these features are already implemented in our prototype tool, while the others are planned as future work.

IV. Data visualisation in CodeMetropolis

CodeMetropolis is a command line tool written in C# and uses the Substrate [**substrate-website**] library for .NET Framework. It takes the output graph of Columbus Tool [**Columbus**] and creates a Minecraft world from it. Columbus Tools area collection of various programs, which are able to analyse and measure static artifacts related to the source code. 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 levels. The data level contains the various object and their data, which are directly related to the measured artifacts, for example classes. On the other hand, metaphor level is build up from the visual representations of these, for example buildings and floors. On the data level, each entity has its own property set – for example metrics, which are displayed on the metaphor level. The buildings in our metropolis are parts of this metaphor, and they have a couple of attributes which control visual appearance. The items which are highlighted on the graphic in Figure 3 represent various source code entities, while the properties are mapped to the attributes in order to visualise the data. In this concrete case you see a couple of namespaces visualised as stone plates. They are contain two classes represented by buildings, finally these have several methods, which size and complexity are mapped to the width and height of the floors.

Figure 4 shows a concrete class from JUnit represented with a single building on a plate. In this example source

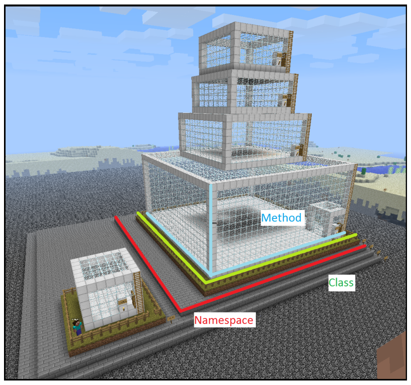

Figure 3. Items of the metaphor level

code metrics complexity (McCC) and size (lLOC) are used as attributes. The metrics have been normalized to scale between a minimum (4 blocks) and a maximum (25 blocks) value.

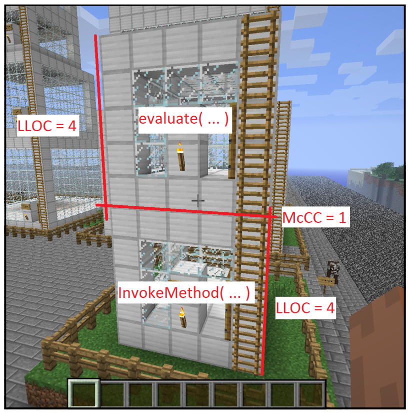

Figure 4. Example class visualization

V. Benefits and further possibilities

The usefulness of the metaphor of CodeMetropolis depends on various factors including the experience and personal values of the users. For 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 further plans and ideas will be explained.

A. Metaphor related ideas

1) Understanding inter-metrical relations: 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. 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 perceive the represented source code as a whole.

2) 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.

B. Help system

1) 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.

2) In-game explanations: Posts, wall signs and books will be used to explain the meaning of the various attributes and to show the source code of the corresponding element.

C. Collaboration and code management

1) Inter-user communication: Besides the traditional forms of communication like textual chat and audio connection users will be able to use in-game items and techniques to interact with each other. For example, they could leave signs as notification or if they are walking together in the virtual world they could simply point to or go to a specific part of the “code”.

2) Round-trip source code management: The changes between the source code and the metropolis will be propagated to each other.

3) Visualize source code history: The functionality of open-source multi-player servers will be extended to be able to visualise 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.

4) Annotating entities: When developers are inspecting the source code, they could leave comments to mark its parts. A future version of our conversion tool will support code annotation. When developers put a wall or post a sign 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 for instance.

5) 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 control system. With the use of historical data, representations can be generated for each revision and then several computer controlled players could be used to literally build these parts of the metropolis. Each such player could represent 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.

VI. Conclusion

Classical visualisation techniques are proved to be useful in many situations but they fail to maintain the motivation of developers in some circumstances. The provided metropolis metaphor, combined with high quality graphical techniques and advanced collaborative features of today’s computer games, has enough expressive power to represent the complex items of the source code and hopefully maintain motivation. In our opinion, software development could be made more interesting and motivating if we unite the solid engineering practice and technologies from the industrial segment with the endless fantasy and joy of creation found in games. As one of our developers said: “It makes software metrics such fun that you want to do it.”

We created a proof of concept implementation for this metaphor. The current prototype implements various basic functionalities, but the more advanced collaborative features overviewed above will be implemented in the future. Eventually, we want to offer a really useful tool in the future not only for enthusiastic developers who are spare time gamers but also for fulltime developers and managers in the software industry.

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